An Empirical Examination of Interdisciplinary Collaboration within the Practice of Localisation and Development of International Software

Malte Ressin

A thesis submitted in partial fulfilment of the requirements of The University of West London for the degree of Doctor of Philosophy

October 2015

In 1923, Austrian author Felix Salten wrote a novel about a buck's life in the woods, experiencing from birth the ruthlessness of nature, intrusion of man, maturation into adulthood, and finally ascension to solitary leader. The popular book, a pitiless commentary on the relationship between nature and humans, was soon picked up by a young cartoon producer to be made into an animated movie for children. Walt Disney's Bambi premiered in cinemas in August 1942.

In the original novel, written in German, Bambi is a roe deer or "Reh", a species only occurring in Europe. Because Disney assumed that the US-American audience would relate better to a local species, Bambi was changed into a white-tailed deer or "Weißwedelhirsch", or short "Hirsch". It is a minor change since both species are related, but they are visually distinct: Roe deer have no tail, and more importantly, their bucks do not grow majestic and impressive antlers like most other deer.

When Disney's Bambi reached German cinemas in December 1950, the dubbing translators had kept closely to Salten's book. At Bambi's birth, his mother's species is identified as "Reh". But because the visuals remained unchanged, once Bambi grows up and meets his father, their antlers clearly show both to be "Hirsche".

Thus, a generation of German children grew up with the so-called "Bambi-Irrtum" or Bambi error: the understanding that "das Reh ist die Frau vom Hirsch", that instead of being a distinct species, roe deer are female deer.

Abstract

Acceptance on international markets is an important selling proposition for software products and a key to new markets. The adaptation of software products for specific markets is called software localisation. Practitioner reports and research suggests that activities of developers and translators do not mesh seamlessly, leading to problems such as disproportionate cost, lack of quality, and delayed product release. Yet, there is little research on localisation as a comprehensive activity and its human factors.

This thesis examines how software localisation is handled in practice, how the localisation process is integrated into development, and how software developers and localisers work individually and collaboratively on international software. The research aims to understand how localisation issues around the above-mentioned classifications of cost, quality and time issues are caused. Qualitative and quantitative data is gathered through semi-structured interviews and an online survey. The interviews focused on the individual experiences of localisation and development professionals in a range of relevant roles. The online survey measured cultural competence, attitude towards and self-efficacy in localisation, and properties of localisation projects. Interviews were conducted and analysed following Straussian Grounded Theory. The survey was statistically analysed to test a number of hypotheses regarding differences between localisers and developers, as well as relationships between project properties and software quality.

Results suggest gaps in knowledge, procedure and motivation between developers and translators, as well as a lack of cross-disciplinary knowledge and coordination. Further, a grounded theory of interdisciplinary collaboration in software localisation explains how collaboration strategies and conflicts reciprocally affect each other and are affected by external influences. A number of statistically significant differences between developers and localisers and the relevance of certain project properties to localisation were confirmed. The findings give new insights into interdisciplinary issues in the development of international software and suggest new ways to handle interdisciplinary collaboration in general.

Table of Contents

List of Abbreviations and Acronyms

ACM:	Association for Computer Machinery
ACT:	Attitude towards Computer Technology
ANOVA:	Analysis of Variance
APA:	American Psychological Association
API:	Application Programming Interface
ASCII:	American Standard Code for Information Interchange
ASTTI:	Association Suisse des Traducteurs, Terminologues et Interprètes
ATL:	Attitude Towards Localisation
BSA:	British Sociological Association
CAT:	Computer-Assisted Translation
CQ:	Cultural Intelligence
CQS:	Cultural Intelligence Scale
EU:	European Union
GALA:	Globalization and Localization Association
GILT:	Globalisation Internationalisation Localisation Translation
GMX:	Global information management Metrics eXchange
GT:	Grounded Theory
G11N:	Globalisation
HCI:	Human-Computer Interaction
HSD:	Honest Significant Difference
ICAPS:	Intercultural Adjustment Potential Scale
IEEE:	Institute of Electrical and Electronics Engineers
IETF:	Internet Engineering Task Force
IP:	Internet Protocol
ISO:	International Organization for Standardization
ITS:	Internationalization Tag Set
I18N:	Internationalisation
LE:	Localisation Effort
LISA:	Localization Industry Standards Association
L10N:	Localisation

LSP:	Localisation Service Provider
M:	Mean
MFC:	Microsoft Foundation Classes
MT:	Machine Translation
OS:	Operating System
RQ:	Research Question
SD:	Standard Deviation
SEL:	Self-Efficacy in Localisation
SEU:	Self-Efficacy in Usability
SPSS:	Statistical Package for the Social Sciences
SRX:	Segmentation Rules eXchange
TBX:	Term Base eXchange
TM:	Translation Memory
TMS:	Translation Memory System
TMX:	Translation Memory eXchange
T9N:	Translation
UCS:	Universal Character Set
UI:	User Interface
UTF:	Unicode Transformation Format
UWL:	University of West London
UX:	User Experience
VoIP:	Voice-over-IP
WPF:	Windows Presentation Foundation
XAML:	Extensible Application Markup Language
XLIFF:	XML Localisation Interchange File Format
XML:	Extensible Markup Language
XP:	Extreme Programming

List of Tables

List of Figures

Chapter 1 Introduction

Software localisation is an important aspect for internationally published software. This research aims to research the causes of problems in software localisation and the adaptation of software for international markets. In particular, it examines how localisation budget, quality and schedule is influenced by developer-translator collaboration, development, localisation infrastructure and processes.

1.1 Software Localisation

International software is software to be used in different countries. Such software needs to be adapted to the target markets' languages and cultures in order to ensure usability and acceptance. Failure to consider the cultural background of users disrupts communication between software and user, and threatens eventual use. Successful adaptation, on the other hand, increases the software's effectiveness and efficiency. Localisation plays a crucial role in spreading software across cultural boundaries and is therefore understood as both a business objective to successfully enter new markets, and inclusion counteracting the digital divide by allowing smaller and underprivileged cultures access to information technology.

The adaptation of a product for specific markets is called localisation. Localisation usually includes translation of text, changing units and symbols for currencies and measurements, and modifying the formats for displays of time or other measures. More advanced adaptations extend to colours, layout, functionality and provisions for technical infrastructure, or may go as far as catering for different business models. Localisation is usually conducted by translators. It is often separately performed on an otherwise finished product.

Closely related to localisation is the process of internationalisation. This is the activity of separating culturally dependant and independent parts of the software by extracting or removing all cultural references. This creates a culturally neutral version that can be configured to specific cultures. Internationalisation is usually done by software engineers.

Originally, just as software was limited by hardware constraints, it was also constrained by culture when it was created in Western countries for a Western audience. Thus, many fundamentals in computing were strongly influenced by Western culture. This manifested

itself in lack of support for conventions such as non-Latin scripts and right-to-left languages. Considerable effort was undertaken to overcome many of these early restrictions, for example through the creation of the Unicode standard so that software would support any script in the world. Accordingly, the tasks, abilities and concerns of software localisation today are fundamentally different from those 20 years ago.

The importance of software localisation continues to increase because, as the global proliferation of computers and adoption of the World Wide Web continues, users in new regions gain access. Accordingly, more localisation is required. In addition, software is now developed in countries other than the Western world, which in turn becomes a localisation target.

The scope of localisation increased as well. Software use was originally restricted to very few professional groups. But as the advent of desktop computing and the personal computer proliferated software to non-technical and untrained users, the notion of usability and user experience (UX) have gained importance in the design of human-computer interaction (HCI), of which culture has become an important aspect.

Similarly, one of the premises of HCI used to be that human use of computers is conscious, visible, and can be designed for. But this is changing as computers and software become more and more ubiquitous and pervasive. As computing areas such as mobile devices, social computing, augmented reality and robotics spread, interfaces diversify, take on new modes, become ephemeral and sometimes disappear altogether. This switch from interface-driven to interaction-driven use necessitates stronger consideration of users' behaviour, values and expectations, and localisation has to move beyond the presentation layer to include all aspects of software.

1.2 Current Challenges in Software Localisation

Challenges in localisation have been identified as cost, quality, and time, i.e. the effort required, impact of localisation on the product, and the delay it causes to development completion.

Software localisation is considered a comparatively expensive undertaking. Exact numbers are notoriously difficult to obtain and obviously depend on a number of factors such as localisation provider, content volume and number of languages. Hall (2002)

estimates localisation costs to make up around 10% of total development costs. A number of examples are discussed in Collins (2001), bracketing typical localisation costs between $50,000 and $300,000 per language, with staff costs often including additional engineering.

Localisation also takes time. In particular, the time needed to localise source content, but also to handle localisation-related bugs. The practice of *simshipping*, i.e. of releasing international software simultaneously in many markets, curtails available time for localisation even further.

Two other challenges in localisation have been mentioned in the literature: volume, i.e. handling large amounts of content into ever more languages, and access, i.e. providing localisation relatively quickly and cheaply for content which otherwise would not be localised. Extreme cases of cost and time require immediate or near-instantaneous localisation at no, or virtually no, expense. For example, messages on online social networks or customer support documents are often processed through machine translation, crowdsourcing, or a combination of the two.

Shortcomings in localisation practice are a lack of defined processes, an incomplete understanding of localisation activities, and collaboration issues between software engineering and localisation.

It has further been noted that localisation is often approached from a very technical point of view as something that can be parameterised and isolated, focusing on an interface-driven approach to software localisation instead of designing for cultures.

1.3 The Problem Statement

This research started as a puzzle from my experiences as localisation team leader in a mid-sized software company: I had observed that most time was spent on handling localisation issues that had been created by trivial causes, and these causes defied any attempts at proactive prevention. Our despair usually came in the form of if-only, for example:

- If only software engineers finalised user interface (UI) text a month before product release, there would be no translation-caused release delays.

- If only UI designers remembered to leave at least 30% buffer space for translation-expanded text, there would be fewer instances of cut text in the UI.
- If only translators referred to the terminology when translating, we would have fewer retranslations.

Comparing notes with colleagues at other companies confirmed that similar issues exist despite organisational differences, suggesting that the difficulty experienced in localisation might originate in the way localisation is conducted in the context of software development. Process-related shortcomings in localisation practice have further been acknowledged in the literature, e.g. a lack of standard processes (Abufardeh and Magel, 2008b), an incomplete understanding of localisation activities and workflow (Lenker *et al.*, 2011), and issues of collaboration between software engineering and localisation (Abufardeh and Magel, 2010; Lewis *et al.*, 2009). Accordingly, there have been calls to examine the collaboration of software engineering and localisation (O'Sullivan, 2001a; Collins, 2001).

1.4 Aims and Objectives

A review of research and literature around software localisation[1] will establish the following:

1. The development of global software is an effort involving the two disciplines translation and engineering. Generally, these activities are separated into internationalisation and localisation.
2. Most research on software localisation examines internationalisation, localisation, software engineering and translation in isolation. Further, most research is focussed on an isolated aspect in the context of localisation such as the influence of culture on UIs, or the evaluation of technological aids such as translation tools and APIs.
3. There is comparatively little research on the practice of software localisation and the causes of localisation issues.

Conversely, the original research problem, i.e. what makes software localisation difficult and what shapes the contributions of individual disciplines, is narrowed down: first, an

[1] See chapter 2.

empirical examination of software localisation as a whole as opposed to its individual constituents. Second, an examination of the collaboration between developers and localisers. And third, identifying the causes of localisation issues.

The literature review will show that technological developments have been employed to address issues in software localisation. For example, Unicode simplifies the use of any script in software, Localisation APIs trivialise internationalisation of many common software functionalities and UI elements, and CAT tools facilitate collaboration of multiple translators while lowering cost and turnaround time.

However, the sociological aspects of software localisation are left unexplored, specifically how development and localisation professionals work on software localisation, and how they work with each other. Both existing research (e.g. Collins, 2001; O'Sullivan, 2001a; Lewis *et al.*, 2009; Abufardeh and Magel, 2010) and my own experience suggests that localisation issues are caused by this cooperation. Understanding underlying causes and effects in internationalisation and localisation activities might go a long way towards avoiding them.

The literature review will further suggest that project properties have an influence on localisation. Such properties might be use of localisation tools, choice of development models, relationship between developers and customers, and user feedback. Similarly, the distinctiveness of developers and localisers is suggested to be of potential importance, e.g. shared or distinct mental models and so on. Accordingly, the research aims are explained as follows:

- To understand the reciprocal influences between engineering and localisation processes.
- To understand how localisation issues are caused during cooperative work of software engineers and translators.
- To understand the distinctness of developers and localisers relevant to localisation.
- To understand the relevance of cultural competence in software localisation.
- To understand the influence of project and product properties on localisation quality.

This leads to the following research objectives:

1. Analyse accounts about localisation practice, in particular regarding the cooperation of developers and localisers.
2. Examine the role of human factors in localisation as a process.
3. Examine differences of cultural competence between developers and localisers.
4. Determine the influence of project properties on localisation.

1.5 Research Questions

Two different kinds of research objectives are becoming apparent. Research objective 1 and 2 aim at describing and exploring a particular situation. Research objectives 3 and 4 operate within suggested frameworks, here human factors, cultural competence, and project properties. Accordingly, exploratory research is appropriate for objective 1 and 2, while explanatory research is appropriate for objectives 2 to 4.

1.5.1 Empirical Study of Software Localisation

To complete the first research objective, research into the work of engineers and translators seems in order. An examination of how engineers and translators work individually and collectively during internationalisation and localisation leads to a large host of specific questions: What activities do engineers and translators conduct for internationalisation and localisation? How do they conduct these activities? What influences how they conduct them? How do engineers and translators communicate? What do they communicate about? What factors influence what they communicate about, and what not? This is summarised into two research questions (RQs):

RQ 1: How is localisation conducted individually and collaboratively by developers and localisers, and how does this shape each discipline's activities?

RQ 2: How are issues caused during localisation and internationalisation?

1.5.2 Human Factors in Developer-Translator Collaboration

In the literature review, distinctness of developers and localisers and localisation issues as a potential consequence is discussed, in particular regarding cross-disciplinary knowledge (Bauer and Rodrigo, 2004; Russo and Boor, 1993; Sikes, 2011) and collaboration (O'Sullivan, 1989; Honkela *et al.*, 1997). Law (2003) sums this up as human factors

affecting developer-translator collaboration. Next, it will be discussed what human factors are relevant and to what research questions they lead.

For one, there is the distinctness of cultural competence. Its importance for developers has been repeatedly stated, both explicitly (Abufardeh and Magel, 2008b; Ryan *et al.*, 2009; Immonen and Sajaniemi, 2003a) and implicitly (Law, 2003; Abufardeh and Magel, 2009; Mahemoff and Johnston, 1998; Abufardeh and Magel, 2010; Smith *et al.*, 2007; Carey, 1998; Hogan *et al.*, 2004; Liem *et al.*, 2011). On the other hand, the argument is made that because software development is technical, handling of culture is not at the core of software development business (Linna and Jaakkola, 2010). It remains to be seen whether there actually is a difference in cultural competence between developers and localisers.

Related topics are localisation scope and localisation requirements (Giammarresi, 2011; Kalliomäki *et al.*, 1997), the assessment of which is based on an understanding of culture (see e.g. Hoft, 1996). Hence, if indeed there is a cultural competence gradient between developers and localisers, this might lead to different assessments of localisation scope.

Cultural competence and attitude towards localisation effectively are about the relationship developers have with the localisation discipline. However, the relationship *vice versa* is just as important (Law, 2003; Immonen and Sajaniemi, 2003a), particularly whether localisers feel able to handle technical aspects of internationalisation. Cultural competence is a very central consideration in localisation. It has been suggested that cultural skills are affected by nationality or language skills (e.g. Carey, 1998).

Some authors have suggested that, along the lines of a focus on technology, software developers are not favouringly predisposed towards software localisation (Honkela *et al.*, 1997; Sikes, 2011). Developers might not even feel responsible towards the outcome of localisation. In other words, there might be an attitude gradient between developers and localisers.

The literature review presents the project management triangle of cost, quality and time. Some authors have suggested that cultural requirements are often sacrificed during development due to time and budget shortage (Tuffley, 2003; Dunne, 2011), just like software projects in general seem to prioritize cost and time over quality (Boehm, 2006,

2011; Blackburn *et al.*, 1996). Similarly, the concerns regarding product quality might differ between developers and localisers (Abufardeh and Magel, 2009).

In other words, the previous considerations suggest to examine the distinctness of developers and localisers regarding cultural competence, attitude towards localisation, and assessment of software quality and priorities in software development. This leads to:

RQ 3: In what regards are developers and localisers distinct?

1.5.3 Project Properties and Localisation

In the literature review, the three localisation factors cost, quality and time are introduced as relevant properties in which localisation success can be assessed. Besides the individual and collaborative work of developers and localisers, these factors are affected by company culture and established practices, available resources, or market conditions (Giammarresi, 2011). More specific influences have been postulated for specific project properties, most prominently type of software (e.g. Abufardeh and Magel, 2010; Hall, 2000; Giammarresi, 2011), type of user (e.g. Liu and Zhang, 2011), relationship between customer and user (e.g. Honkela *et al.*, 1997; DePalma, 2006), number of target locales (e.g. Ryan *et al.*, 2009), and influence of the software development model (e.g. Fissgus and Seewald-Heg, 2005; Abufardeh and Magel, 2010). Further, there are both expressed and implied suggestions that the localisation outcome can be related to the commercial character of a project, i.e. whether it is a commercial project or not (e.g. Wolff, 2006; Exton *et al.*, 2010).

The aim is to determine how these project properties affect the three localisation factors cost, quality and time. These are inherently difficult to measure, as discussed in section 2.5. Localisation expenses are often not tracked (DePalma, 2006), and localisation quality lacks standardisation (e.g. Tarquini *et al.*, 2010). Hence, localisation effort is chosen as dependent variable[2], leading to the following research question:

RQ 4: What dependencies exist between localisation effort and development project properties?

[2] A detailed operationalisation of localisation effort will be discussed in subsection 3.3.2.

In all, four research questions were identified. The methods of answering them are discussed in chapter 3.

1.6 Research Contributions

The thesis contributes to an understanding of how international software can be developed efficiently. It describes software development and localisation practice used by software engineers, translators, and their managers. Further, it describes the origins of these practices in each discipline's underlying practices, objectives and agendas, as well as the interdisciplinary conflicts arising from them. A grounded theory of interdisciplinary collaboration during software localisation explains how external influences based on general and discipline-specific success criteria, tools and processes provoke different strategies employed by, and cause conflicts between, localisers and developers during the facilitation of interdisciplinary collaboration. Further, the research contributes evidence of gradients in cultural competence and attitude towards localisation between developers and localisers, and of the relationship between localisation and certain project properties.

1.7 Thesis Structure

The thesis is structured into six chapters[3]:

1.7.1 Chapter 2: Software Localisation and Internationalisation

In the second chapter, existing literature is reviewed. First, the meaning of culture is discussed and the key terms *locale*, *localisation*, *internationalisation*, *globalisation* and *translation* are defined. Then, existing research in software localisation and related areas is discussed. The literature review will show that software localisation is an activity involving many disciplines, including engineering and translation, that most existing research examines localisation scope, activities, and their context in isolation, that there

[3] The thesis structure is loosely based on the structure for social science research reports suggested by Wisker (2008). Writing was endeavoured to be gender-neutral throughout the thesis. In order to obscure interviewees' identities, they are consistently referred to using the masculine form. In quotes, ellipses without square brackets ("…") indicate a pause. Occasional modifications and shortenings for clarity are indicated by square brackets. Ellipses in square brackets ("[…]") indicate the omission of words, sentences, or paragraphs. Typeset, layout and referencing style follow The University of West London (UWL) thesis style regulations (UWL, 2015). Presentation of statistical results are based on the guidelines of the American Psychological Association (APA) (APA, 2009).

is relatively little research on the causes of localisation issues in general, and that there have been few comprehensive studies of the practice of software localisation.

1.7.2 Chapter 3: Research Methodology and Method

Based on the research questions and the literature review, two independent studies are constructed. The research aims are to understand the reciprocal influences between software engineering and localisation, the origins of localisation issues in collaborative work, the distinctness of developers and localisers, and the relevance of cultural competence for software localisation. An interview case study using Grounded Theory (GT) will explore localisation professionals' perceptions of social processes, human interactions and organisational contexts involved in the development and localisation of international software. A survey study will test a number of hypotheses about the distribution of cultural competence, self-efficacy, opinions and attitude about localisation in professional localisation roles, and localisation projects and their properties.

1.7.3 Chapter 4: Qualitative Results

The qualitative research shows that the main concern of interviewees is the facilitation of interdisciplinary collaboration between linguistic and technical professionals, determined by constraints, conflicts and the chosen collaboration strategies, which in turn influence each other. The behaviour of developers and localisers can be characterised through a dominance of engineering considerations and processes, and a self-serving behaviour of localisers characterised by the theory of agency.

1.7.4 Chapter 5: Quantitative Results

The quantitative research confirms some of the assumed relationships. Compared to localisers, developers score lower on both cultural competence and attitude towards localisation, but this does not translate into a difference of localisation scope assessment or overall quality criterion prioritisation. Localisation is affected by the existence of a development model and certain software types, and the more locales are targeted, the more effort is expended. However, user type, relationship to the user, and commercial nature of a software project do not affect localisation.

1.7.5 Chapter 6: Conclusion

The limitations of the qualitative and quantitative approaches and analysis methods are discussed. Further, the research contributions are discussed and their potential implications for practice elaborated. Potential for future research is explored.

Chapter 2 Software Localisation and Internationalisation

In the first chapter, software localisation was introduced as the adaptation of software for use in international markets in order to foster acceptance and use of increasingly pervasive and relevant software, both as a business and inclusiveness objective. The introduction also set the original problem statement: to examine how localisation issues are caused by the process of software localisation in the context of software development.

This chapter introduces research and development with relevance to software localisation, software internationalisation, processes and work steps, and limitations on them. It discusses the nature of localisation and influences on the process and the work of developers and translators, including localisation and internationalisation requirements, bugs, tools and utilities, standards, rules and guidelines, organisational forms and existing empirical research. Sections 2.1 details the literature search strategy. Sections 2.2 and 2.3 introduce relevant terminology and explore seminal research on culture, cultural differences, and the consequences for software products published in international markets. Sections 2.4 and 2.5 review scope of localisation, the localisation factors cost, quality and time, and localisation issues, i.e. bugs and process difficulties. Section 2.6 explores how research and practice tackle internationalisation and localisation through technical and procedural means. Section 2.7 discusses existing research on software localisation practice and contrasts it with existing empirical research on causes of localisation issues. The following will be shown:

First, the adaptation of software for international markets is a multidisciplinary activity.

Second, the majority of software localisation research is located around the context of software localisation, i.e. evaluation and examination of activities and tools used during translation, internationalisation and localisation.

Third, there is comparatively little research about the practice of software localisation and internationalisation in the context of software development, and this existing research suggests that the practice and engineer-translator cooperation are in need of improvement.

Fourth, while there is a good amount of research and development toward improving localisation, there is comparatively little research examining the causes of issues, and research that does is often limited to specific issues or narrows down to a particular localisation aspect.

The findings from this chapter narrow down the problem statement to original research aims and objectives, and eventually the research questions as discussed in the introduction, leading to the choice of research methodologies in chapter 4 to examine the activity of software localisation.

2.1 Literature Search Strategy

For this literature review, publications were perused when they addressed in any way definitions and motivation for localisation, human factors in software localisation and work practice of localisers and developers working on international software, including tools, utilities, standards and processes.

Because there are competing meanings for the terms localisation, language, translation and internationalisation, a search in common literature search databases led to a lot of false positives. Instead, this literature review started by a systematic manual search through the most relevant publications, as well as some websites listing peer-reviewed publications on localisation and related topics. Appendix G lists these resources.

2.2 From Culture to Locale

The activity of software localisation originates from the requirement to adapt software. The more complex software becomes, the more likely it is that it is not globally applicable to all markets. Barber and Badre (1998) explicate this by observing that there is no such thing as one global interface suitable for all cultures. And indeed, *culture* is the term under which idiosyncrasies of different software markets are often subsumed, with a market's specific requirements referred to as *cultural differences*. To understand how culture is tackled during the development of international software, the term and applicable definitions in science and software development are discussed.

The meaning of the term *culture* is complex. In general use, culture can refer to, among others, achievements in the field of art, individual sophistication, tradition and mores, or accepted norms and expected behaviour in professional or social groups. It is an

ambiguous and flexible term (del Galdo, 1996) used frequently in academia and everyday life, yet with many meanings. Kroeber *et al.* (1952) counted more than 156 different definitions, none of which are generally agreed (Kamppuri, 2011). The ambiguity of the term *culture* has been commented on by many researchers in computing (Smith *et al.*, 2007; Goggins and Mascaro, 2011; Kamppuri, 2011).

How culture is handled in production environments is significantly shaped by how culture is perceived (Sun, 2002; Sturm, 2002). For this reason, it is important to discuss views on culture expressed in software development.

Rauterberg (2006) explains culture as the integration of human behaviour, attitudes, norms, values, beliefs, actions, and communications in ethnic, religious or social groups. In other words, culture can be understood as attitudes, beliefs and behaviours of a group, including special interest groups such as religion, race, society, organisation, nationality, history, language or level of technical sophistications (del Galdo, 1996; Kamppuri, 2011; Linna and Jaakkola, 2010). Cultural values have been linked to biological and social factors (Ito and Nakakoji, 1996; Kamppuri, 2011).

In the context of HCI, culture is a cognitive phenomenon affecting artefacts and behaviour (Kamppuri, 2011). It is located on or close to the level of language and national culture, as opposed to e.g. culture of smaller groups (Clemmensen and Roese, 2010), although the notion of national culture is problematic as well (Smith *et al.*, 2007) as many nations[4] are comprised of various ethnic groups differing in languages, traditions, norms, etc. India, Russia and the USA are examples of nations that are neither linguistically nor culturally homogeneous. But because a language can subtly differ between different nations as well, e.g. in the case of British English, US English and Australian English, culture cannot be equated with language either (Abufardeh, 2008, pp.9, 10).

2.2.1 Cultural Models

Culture has been examined in science through *cultural models*. These models specify so-called *cultural dimensions*, also called *international variables* (Hoft, 1996). The notion is based on research by Edward T. Hall and suggests to find behaviour, properties or

[4] The term *nation* is mostly applied either socio-culturally, referring to collectives of people with common characteristics such as language, culture and ethnicity, or geopolitically, referring to country states. In this thesis, it is used in the latter way for consistency with the terms *national* and *international*.

artefacts that consistently differ across cultures. Hall developed a number of cultural dimensions from his field work in Europe, the Middle East, Asia, and with indigenous cultures in North America, for example to what degree messages are context-dependant, and whether time is perceived as monochronic, i.e. sequential, or polychromic, i.e. parallel[5] (Hall, 1959, 1966, 1977). Hall asserts that cultures can be described by their position on a spectrum of a number of such cultural dimensions. His conclusions have been criticised for being based on observations on qualitative insights on the level of larger geographical areas, e.g. the Eastern Mediterranean, Western or Northern Europe that are not necessarily culturally homogeneous (Ahmed *et al.*, 2008).

A number of cultural models exist and have been categorised into four meta models (Hoft, 1996; Linna and Jaakkola, 2010). The *objective vs. subjective model* distinguishes between objective culture and subjective culture, i.e. between tangible and visible culture such as artefacts, behaviour, and organisation on one side, and values, norms and other psychological features on the other side. The *iceberg model* views the subconscious aspects of culture such as values and beliefs as an iceberg's large underwater body. On this is built the comparatively small visible part of the iceberg above the surface, e.g. artefacts and behaviour. The *pyramid model* shows culture as middle layer and linking element of human nature, which is common for all humans and therefore serves as the pyramid's base, and personality, which is individual and is at the pyramid's stop. The *onion model* describes culture as consisting of different layers, where the outer layers represent artefacts and behaviour, and the layers further inward stand for norms, values and beliefs.

Cultural models have been used to inform the adaptation of software for different cultures (e.g. Hoft, 1996), and in HCI and behavioural research both as source for hypotheses towards the examination of cultural fit of UIs (e.g. Hall, 2000), and to explain findings (Kamppuri, 2011). On the other hand, cultural models have been described as encouraging stereotypes and generalisation (Kamppuri, 2011) and applying an unsuitable unit of analysis, i.e. nations or larger regions (Hua *et al.*, 2014).

[5] This is also referred to as M-time and P-time.

Some research uses Trompenaars' model of national cultural differences (Trompenaars and Hampden-Turner, 1998)[6], developed from a survey of 8,841 managers in 43 organisations. It considers culture as the problem solving strategies of groups, in particular in the context of business management. For this reason, its application outside of this context has been criticised (e.g. Hoft, 1996).

2.2.1.1 Hofstede's Cultural Dimensions Model

Hofstede's Cultural Dimensions Model (Hofstede and Hofstede, 2005) is arguably the most popular cultural model. Hofstede considers culture as a group-discrete cognitive programming he calls *software of the mind*, e.g. norms, expectations, concepts, and responses to the environment which people acquire throughout their life from their social environment. Through empirical research on ca. 116,000 employees at IBM in the late 1960s and early 1970s, six linearly independent dimensions of cultures were determined:

- Power distance: Does social hierarchy form relationships?
- Individualism vs. collectivism: Are people individuals or part of a group?
- Masculinity vs. femininity: Is society oriented towards either male values such as power and wealth, or female values such as empathy and friendship?
- Uncertainty avoidance: Are future events perceived as controllable?
- Long-term vs. short-term orientation: Is motivation gained through immediate rewards or future payoffs?
- Indulgence vs. self-restraint: Do social norms control self-gratification?

Due to it being one of the first models to tackle the phenomenon of culture in a quantitative way through comparatively straightforward dimensions, Hofstede's model has become ubiquitous and dominant wherever culture is supposed to make an impact. It has been widely adapted in many fields from politics via science to economy and

[6] Trompenaars describes the following cultural dimensions:
- Universalism vs. particularism: Do morals and judgement follow rules or relationships?
- Individualism vs. communitarianism: Are people perceived as individuals or as part of a group?
- Neutral vs. emotional: Are emotions subdued or expressed freely?
- Specific vs. diffuse: Are private and business life separated?
- Achievement vs. ascription: Is status gained through achievements or through titles?
- Sequential vs. synchronic: Is time perceived as coherently linear or circular, or as incoherent?
- Internal vs. external control: Is our environment perceived as controllable, or does it control us?

continuously tested against new data, being augmented where necessary[7]. For the same reasons, it has been extensively scrutinized, and various methodological issues were criticised (see Kamppuri, 2011; McSweeney, 2002). Among others, Hoft (1996) criticized that the questionnaire employed was not culture neutral, and McSweeney (2002) considered Hofstede's approach to determining cultural dimensions through survey difference analysis tautological. Abufardeh and Magel (2010) pointed out multiple selection biases in Hofstede's data, which was almost exclusively gathered at one specific company and therefore reflects more on organisational than national culture. Further, Hofstede did not present results consistently on a national level. While results were itemised for many countries, some nations were merged into culturally inhomogeneous conglomerates such as the Arab World (see Ahmed *et al.*, 2008). Further, the model's indications do not always match up with observation, as e.g. noted by Hall *et al.* (2009) regarding a perceived reluctance of organisations to make decisions in a society which is assumed to have low uncertainty avoidance. Similarly, Kamppuri (2011) reports interactions in Tanzania and Finland that frequently did not match Hofstede's model.

This might hint towards misunderstanding of the actual applicability of cultural models (Kamppuri, 2011): On one hand, Hofstede appears to encourage the application of his model, particularly in the sector of technology. On the other hand, Hofstede differentiates between cultural values, i.e. preferences as described by his cultural dimensions, and cultural practices, which are not directly described by his model. Hofstede argues that values are more stable than practices, but it does question the model's use when designing products for different cultures.

2.3 GILT

Now that user groups requiring culturally adapted software versions can be specified through locales, the notion of *software localisation* can be discussed, that is, the adaptation of software for users in different locales. Localisation is part of the so-called GILT[8] framework standing for Globalisation, Internationalisation, Localisation,

[7] Originally, Hofstede's model consisted of four dimensions. Long-term vs. short-term orientation was added in the second edition in 2001, indulgence vs. self-restraint was added in the third edition in 2010.
[8] Dunne (2006) suggests to refer to it as TLIG because practitioners build awareness in that order.

Translation[9] (Anastasiou, 2009; Yuste, 2005). Although these terms, particularly the first three, are commonly used, there is only a rough consensus on their meaning and relation to each other (Schäler, 2007; Dunne, 2006). In short, translation refers to the transfer of text from one language to another, localisation is the adaptation of a product to a specific locale, internationalisation is the activity of preparing a product for adaptation to specific locales, and globalisation stands for the practice of distributing a product globally. Each of these activities has their own characteristics, rules and pitfalls.

2.3.1 Locale

The ambiguous character of the term culture was mentioned earlier and is reflected in the differing structures of cultural models. This ambiguity is problematic for two reasons:

First, the notion of adapting software for different cultures implies a finite list of cultures to adapt for. But there is no generally agreed upon list of cultures. A similar problem was already touched upon in the previous section on cultural models considering different units, e.g. regions or nations.

It might appear that nations are a very convenient unit to adapt software to, so that for each nation, there is a version. Assuming that there are currently 195 nations[10], these might be mapped on one byte. It might further appear that nations map nicely to certain aspects in which software needs to be adapted, such as law, which generally differs between nations.

However, distinguishing software adaptations on a national level implies that culture is nationally homogeneous. Unfortunately, as was already mentioned when scrutinising Hall's and Hofstede's cultural models earlier, this is not the case and many nations are both culturally and linguistically inhomogeneous. For example, the USA includes population groups with cultural traditions from Europe, Latin America, Africa and Asia, but also Native American traditions. Similarly, many nations have more than one official language, e.g. Switzerland and Canada, or large population groups with their own language, such as the Latino population in the USA. So, by creating nation-specific

[9] Globalisation, Internationalisation, Localisation and Translation are often abbreviated as G11N, I18N, L10N and T9N, with the middle number indicating the count of omitted letters.
[10] At the time of writing, the United Nations (2015a, 2015b) have 193 member states and acknowledge two non-member states.

software, one would in part miss adaptations for cultural groups on a sub-national level, i.e. French speakers in Canada.

Another segregation might be suggested on the level of languages so that for each language there is one version. That, too, brings difficulties due to linguistic and cultural inhomogeneities. For example, the predominantly English countries Australia, New Zealand and Canada use the metric system, whereas the USA and the UK each use their own system of imperial measurements. French is an official language in France, but also in the culturally distinct nations Ivory Coast and Congo.

The second problem of the ambiguity of culture for development of international software is to determine along what dimensions software has to be adapted for each culture.

To solve those two problems, the notion of *locale* has been introduced. A locale defines a set of linguistic, cultural and technical specifications including script, orthography rules, units of measurements, and data presentation formats (Tarquini *et al.*, 2010; Hudson, 1997; Dr. International, 2003; Anastasiou and Morado Vázquez, 2010; Mahemoff and Johnston, 1998; Hall, 2000). A locale is specified through a language-region pair, i.e. the combination of an ISO 639 language code and an ISO 3166-1 country code (ISO TC 37/SC 2, 2002; ISO TC 46, 2013). These locales can be amended by additional information related to sorting instructions, character classifications and formats. For example, en-US refers to the English-speaking US market, en-GB refers to the English-speaking United Kingdom.

2.3.2 Translation

Translation between two languages, also called *interlingua translation* or *translation proper*, is the "interpretation of verbal signs by means of some other language" (Munday, 2009, p.5), more mundanely understood as the transfer of text from a source language to a target language to facilitate communication (Anastasiou and Schäler, 2010; Malmkjaer, 2008). Translation is not done word for word, but meaning for meaning, and therefore applies to more than just words on a page without perfect equivalence (Munday, 2009; Schubert, 2009).

Bauer und Rodrigo (2004) distinguish between *sender-commissioned translation* and *receiver-commissioned translation*. The former, e.g. a website or software for customers, usually demands high quality, whereas the latter, e.g. a social media user reading updates from foreign friends, often accepts less than perfect translations.

The activity of translation depends on what is translated, and translators specialise accordingly. Russo and Boor (1993) makes the case that translating UIs is more difficult than for example literary translation because of the many subtleties included. Translating for software also includes auxiliary activities such as researching information (Schubert, 2009), string management (Hogan *et al.*, 2004), post-editing (Rico and Torrejón, 2012), and even content creation (Yuste, 2005). At times, even tasks closer to software engineering can become part of a translator's activities, e.g. resizing of UI elements to accommodate text expanded during translation (Fissgus and Seewald-Heg, 2005; Hartley, 2009).

2.3.3 Localisation

Localisation is the process of adapting a product for a specific locale by translating or otherwise adapting the relevant locale-dependent content for the benefit of users in said locale (Esselink, 2000; Collins, 2002; Dunne, 2006; Lenker *et al.*, 2011; Liem *et al.*, 2011; Sikes, 2011). It can be seen as a special case of accessibility, i.e. the adaptation of software for diverse user groups with very specific needs, and a way to diversify and reach more users (Perlman, 1999).

Localisation can apply to services and physical wares. The two most common kinds of software localisation deal with applications and websites (Anastasiou, 2009). Its cornerstones are the translation of text and the adaptations to cultural conventions (Anastasiou and Schäler, 2009), but the exact scope of localisation is somewhat deeper and fuzzier and includes consideration of locale-specific data and number formats as well as date and time formats, calendar systems, units of measurement, currency, images, icons and symbols, aesthetics, conventions for names, sound and colour, gender roles, depiction of national borders and geography, locus of control, functionality, UI layout and time zone handling (He *et al.*, 2002; Cyr and Trevor-Smith, 2004; Hall, 2002; Anastasiou and Morado Vázquez, 2010). The scope of localisation will be discussed in section 2.4.

2.3.3.1 The Relationship between Translation and Localisation

Although, as discussed previously, the scope of localisation goes beyond text translation, language quality is a fundamental aspect of successful localisation (Exton *et al.*, 2010; Anastasiou and Schäler, 2009). Software communicates mostly through text (Sikes, 2011). Accordingly, localisation consists most prominently of translation, hence localisation is perceived as translation for software (Dunne, 2006; Sikes, 2011). Insofar, localisation is conceptually related to screen translation, e.g. the translation of movie subtitles, and Chiaro (2009) argues that it might often be the more fitting term when localisation is limited to UI text translation.

Nonetheless, there are different interpretations of the relationship between translation and localisation: Translation and localisation is seen by some as identical, being merely synonyms for the same concept applied in different contexts. Others see in localisation a comprehensive activity of which translation is a part of.

Illustrating the first, Hartley (2009) writes that it has long been accepted in the translator community that localisation is a specialist term used when the concept of translation is applied to software. Hartley is aware of the different scope localisation work requires, e.g. resizing of the UI, but appears to understand translation as the larger concept of adapting a product beyond text translation (see Munday, 2009; Law, 2003). In other words, Hartley finds localisation and translation conceptually identical, and localisation as merely the concept of translation applied to software. A similar view is given by Dohler (1997), who argues that localisation of information technology products is a translation of the whole product, rather than just a product's textual elements like packaging and handbooks.

On the other hand, many authors explicitly understand localisation and translation as not synonymous, arguing that localisation is distinct from translation because it also involves non-textual elements such as layout, icons, colour, and sound (e.g. Anastasiou, 2010b; Anastasiou and Schäler, 2010; Collins, 2002; Sikes, 2011; Hudson, 1997), and that the notion of localisation as software translation does not apply precisely because translations are affected by restrictions such as available space in the UI (Anastasiou and Schäler, 2009). Further, localisation is different from translation because software as a product is less language-centric and more communication- and information-centric

(Fissgus and Seewald-Heg, 2005), and because translated text is only part of a software product, whereas in translation of a book, translation is the whole product (Chiaro, 2009). Following this argument, localisation was different from translation even if it merely focused on UI text.

The relevance of this point is that in commercial software development, localisation is often limited to textual elements in software (e.g. Abufardeh and Magel, 2010), or at best include other superficial presentational elements for replacement, e.g. symbols and graphics (Kamppuri, 2011, p.24). This seems to be a deliberate decision to simplify localisation, rather than a lack of need to adapt the software further, as e.g. reported by Sun (2004b).

2.3.4 Internationalisation

There are two ways of adapting software for locales. A trivial approach is to create copies of the code for each locale and make the required adaptations for each locale in the respective copy. This is referred to as *retrofitting*. Since it creates redundancy and duplication of effort, e.g. if a bug requires code modification in each copy of the code, it is generally considered to be an expensive way of conducting localisation (Kumhyr *et al.*, 1994; Dohler, 1997).

The relatively cheaper approach is to design software in such a way that all locale-dependent aspects can be configured. This is commonly called *software internationalisation* (Caddell and Hall, 2005; Liem *et al.*, 2011; Carey, 1998; Barbour and Yeo, 1997; Hudson, 1997), but has also been referred to as *enabling* (Kumhyr *et al.*, 1994; Hudson, 1997), *design-for-localisation* (Hall, 2002), or *globalisation*[11] (Dr. International, 2003; Dröge *et al.*, 2006).

Software internationalisation can be understood in different ways: Some see it as separation of locale-dependent and locale-independent software elements (e.g. Caddell and Hall, 2005; Carey, 1998), others as developing a culture-neutral software core (e.g.

[11] A different meaning for globalisation as global business strategy is introduced in subsection 2.3.5.

Barbour and Yeo, 1997), yet others as designing software to be configurable[12] for various locales (e.g. Liem *et al.*, 2011; Sikes, 2011). Esselink (2006) suggests it should be all three.

Internationalisation is sometimes implied to be simplification and reduction of ambiguities (Kumhyr *et al.*, 1994) or even the removal of any locale-relevant content so that localisation is not required (e.g. Law, 2003). A large body of advice on how to make a product more culture-neutral can be found in Microsoft Corporation Editorial Style Board (2004). However, the main objective of internationalisation is efficient localisation (Combe, 2011), including localisation maintenance, for example easy resizing of UIs to accommodate for translated text which has changed in length (Tarquini *et al.*, 2010). Further, internationalisation simplifies testing if testing for many locales can be reduced to one test (O'Sullivan *et al.*, 2003).

Internationalisation requires code creation without assumptions of any single locale (He *et al.*, 2002). Sikes (2011) breaks down internationalisation to three task: removal of culture-dependant elements from software design, separation of presentation and application logic in the software architecture, and support of global norms, e.g. character sets and locale-dependent application behaviour.

Software internationalisation can be implemented in three different ways[13] (Carey, 1998; Lehtola *et al.*, 1997): In *compile-time internationalisation*, modified code is created and compiled for each target locale, similar to re-engineering. In *link-time internationalisation*, locale-specific objects and resource files are created during compilation of the code from a common code base. In *run-time internationalisation*, one set of object and resource files are created and locale-specific resources are loaded during software execution (e.g. Exton *et al.*, 2010). The latter approach supports locale switching at runtime, but requires the most computing power due to the needed dynamic link calls (Kokkotos and Spyropoulos, 1997a).

[12] Despite being identical for all intents and purposes, configuration files contain guidelines or commands while resource files contain values (Kokkotos and Spyropoulos, 1997a).

[13] He *et al.* (2002) distinguishes between internationalisation and localisation approaches and lists seven permutations: run-time localisation, compile-time localisation, compile-time internationalisation with compile-time localisation, compile-time internationalisation with link-time localisation, compile-time internationalisation with run-time localisation, design-time internationalisation with link-time localisation, and design-time internationalisation with run-time localisation.

The necessities of the project should dictate the most suitable localisation or internationalisation approach, i.e. whether to re-engineer, internationalise, retrofit, or even reverse-engineer an already compiled product. Depending on whether software needs to be partly internationalised, i.e. supporting only specific locales, or fully internationalised, i.e. supporting any possible locale (Barbour and Yeo, 1997), some approaches are suitable for one or few locales, while others are preferable if software is localised into many locales (He *et al.*, 2002). The required effort is determined by internationalisation scope and method, which are therefore dependent on motivation, i.e. business potential if applicable (Honkela *et al.*, 1997).

There is a consensus that internationalised software products are more efficient in development due to their single code base, and consume fewer time and resources during localisation, leading to cost savings and a faster time to market (He *et al.*, 2002; Carey, 1998). Accordingly, internationalisation practically is a requirement for software localisation (Anastasiou and Schäler, 2009; He *et al.*, 2002; Dunne, 2006). The overall success of a localisation project depends to a major extend on the quality of its internationalisation (Giammarresi, 2011).

Internationalisation can be part of the original development, or may be applied to existing, not yet internationalised software (Hall, 2002; Honkela *et al.*, 1997). The latter is referred to as *re-engineering* (Peng *et al.*, 2009) or *re-enabling* (Mahemoff and Johnston, 1998). Re-engineering already finished software is generally connected with additional effort and expenses (Mahemoff and Johnston, 1998; Kumhyr *et al.*, 1994; O'Sullivan, 2001a). Law (2003) describes the case study of an internet portal where re-engineering existing prototypes cost ca. 1 month. Particular challenges were the refactoring of code to accommodate universal character encoding and the general transformation from static to dynamic UI text. Similarly, the inclusion of right-to-left languages can require extensive reengineering (Giammarresi, 2011). Exton *et al.* (2010) describe the effort to re-engineer an existing application to use their Babel Client Library localisation framework. The authors detail the kinds of difficulty to be expected when retrofitting an existing application and conclude:

> *[A]lthough it is possible to retro fit an existing application with the [Babel Software] architecture it is not advisable and so [Babel Client*

> *Library] should be incorporated into a client application's design during the initial development phase. (Exton* et al.*, 2010, p.46)*

Similarly, when contrasting Japanese and English versions of a website and discussing usability issues, localisation and internationalisation, Tarquini *et al.* (2010) found that websites not developed with internationalisation in mind are not easy to localise.

Accordingly, it is recommended to include internationalisation from the start of design and development (Kumhyr *et al.*, 1994; Hudson, 1997; McConnell, 2004, p.48; Dröge *et al.*, 2006, p.423; Ryan *et al.*, 2009; Tarquini *et al.*, 2010). Initially higher costs of early internationalisation will be compensated by savings in the long run (Collins, 2002; Hudson, 1997), and Hudson *et al.* (1997) discuss the practice of involving the internationalisation department early in product development in order to save time, reduce costs, and foster locale-dependent awareness.

Internationalisation does not only mean engineering effort, but also affects software development in other ways, as examined by Abufardeh and Magel (2009), who found that the impact on software security and performance were the most important. Internationalisation also leads to *scattering*, i.e. code changes affecting multiple classes, and *tangling*, i.e. code changes affecting elements which are also affected by requirements other than internationalisation/localisation. Scattering and tangling suggest that cultural factors go beyond the UI and create functional requirements with implementations throughout the software, thus complicating its development due to the many changes of interrelated parts and hampering reusability, extensibility, and traceability of software artefacts. Abufardeh and Magel (2009) call this *crosscutting concerns* and recommend to integrate the identification of such crosscutting concerns into the software development lifecycle as early as possible to reduce cost and time.

2.3.5 Globalisation

The term *globalisation* is often used in two different ways in the context of software localisation. As mentioned previously, the term is sometimes used synonymously with internationalisation.

Other than that, globalisation refers to transforming many local markets into a single global market in which individuals and businesses operate and compete (Dunne, 2006;

Sikes, 2011; Hudson, 1997). It is effectively a marketing term (Anastasiou, 2009) referring to an overall strategy of a more or less global business presence or services, often but not necessarily including locale-adapted software. For a software vendor, internationalisation and localisation is the implementation of globalisation in order to operate globally (Hartley, 2009).

The topic of globalisation touches on the motivation for localisation. Different objectives are possible. Anastasiou *et al.* (2010) distinguish between ordinary or mainstream globalisation, focused on economic aspects and exclusively driven by term sales, and out-of-the-ordinary globalisation for social, political and cultural aspects, driven by independence and open to the community. Similarly, Ryan *et al.* (2009) distinguishes between informative motivation, i.e. increasing the availability of information or software, for example in the case of the - political - EU localizing content for its member states, and commercial motivation, i.e. increasing business by accessing new markets and attracting new users.

As will be discussed later in detail, software localisation requires considerable effort. In the context of this research and relating to the distinction between informative and commercial motivation discussed above, I would like to argue that this effort is spent in order to satisfy one of two objectives: either culture-centred localisation, i.e. to enable populations, languages and cultures, or business-centred localisation, i.e. localisation to ensure international product proliferation and success. These will be discussed in more detail next.

2.3.5.1 Culture-Centred Localisation

Culture-centred localisation refers to catering for specific cultures, for example to preserve minority cultures that would otherwise fade. For small indigenous cultures, localisation is essential for full participation in technological developments and therefore preserving their active use (Barbour and Yeo, 1997; Hall, 2004; Caddell and Hall, 2005). Speedy localisation is important, as once a new technology has been adopted, users avoid even those changes increasing usability or accommodating their culture (Wolff, 2006; Clemmensen, 2010).

Likewise, Ito and Nakakoji (1996) discusses the suitability of the typewriter metaphor for word processing in Japan[14]. Due to the complexity of Japanese script, typewriters had not been widely used. Instead, Japanese was written in a 20x20 grid of rectangular squares. It is suggested that the import of foreign word processing software based on the typewriter metaphor forestalled the creation of a word processing metaphor more suitable to Japanese writing. Nowadays, of course, word processing based on the typewriter metaphor is the norm in Nippon.

Localisation has also been identified as a prerequisite for development through the provision of information technology to developing countries, bridging the *digital divide* between developing and developed world[15] (Abdelnour-Nocera *et al.*, 2011; Hall, 2002, 2004). However, ideally only localised IT technology should be provided in order to avoid technology rejection due to it being perceived as a power relationship statement (Amichai-Hamburger, 2010). The introduction of foreign culture through new technology, e.g. English as dominant IT language, can also develop a life of its own with severe consequences for local languages (Caddell and Hall, 2005; Wolff, 2006; Hall *et al.*, 2009).

There also appears to be a certain political aspect to localisation. For example, Hall (1998) notes that while Nepali is written in a derivative of the Devanagari script for which a Unicode[16] encoding exists, this is rejected, apparently because Nepal prefers its own encoding for reasons of socio-political differentiation[17]. A further role is played by socio-political agendas, e.g. participants of the Unicode standard encoding process apparently preferring exclusive Unicode code points for their ethnicity's script, rejection the notion of sharing code points with congruent scripts of different ethnicities (Hall *et al.*, 2014; Hall, 2015).

2.3.5.2 Business-Centred Localisation

Software localisation, i.e. the adaptation for different markets, obviously increases a software product's potential for sales (Schäler, 2007; Hartley, 2009; DePalma, 2006).

[14] The Japanese typewriter metaphor is also mentioned by del Galdo (1996), Nardi *et al.* (2011) and Clemmensen (2010).
[15] The term digital divide is also applied to other, non-national minorities. For example, Pérez-Quiñones *et al.* (2005) apply it to women vs. men in information technology.
[16] Unicode is discussed in subsection 2.6.4.1.
[17] Similarly Ali and Kohun (2007) request recognition of Kurdish culture by implementing a Kurdish locale in software development frameworks.

Many software companies conduct the majority of their sales outside of their domestic market (for numbers, see e.g. Tarquini *et al.*, 2010; Hall *et al.*, 2009). Motivations for what Lenker *et al.* (2011) call *enterprise localisation* might be reactive, i.e. to satisfy customer requests, or strategic, i.e. to expand into new markets (Giammarresi, 2011), though market entry itself does not immediately require localisation (DePalma, 2006). Far from being a question of mere profit, del Galdo (1996) and Giammarresi (2011) suggests that in a globalised word and a global market, localisation is not only an option, but a necessity for software development companies to survive. Accordingly, DePalma (2006) found that about a quarter of the examined companies measure their localisation return on investment, but 74% localised because they felt they have to.

In any way, business-centred localisation differs from culture-centred localisation on two important points: First, it is an industrial process with respective constraints and requirements, e.g. minimum turnaround speed and cost (Hall, 2004). Software localisation brings with it significant cost[18], e.g. discussed by Hall (2002) and Collins (2002), which must be justifiable in a business sense (Hoft, 1996; Hua *et al.*, 2014), e.g. if localisation of the product results in enough additional product sales to make up for the additional cost[19] (Sikes, 2011). And second, business-centred localisation has a different focus and different priorities than culture-centred localisation. For example, Dr. International (2003, p.8) lists consistent look, feel, and functionality of software across different locales as objective of localisation and points out that customers might have an expectation for software to be widely identical. In particular, corporate customers prefer localised, yet homogeneous software for reduced effort in technical support and training. Further, the permissible effort in a profit-oriented international software project, including the decision what to internationalise and localise, is obviously limited by expected revenue (Hudson, 1997).

Consistency across different locale versions is at odds with cultural adaptation. For example, comparing to the example given earlier of the unsuitable typewriter metaphor for word processors in Japan, one can see how not using the typewriter metaphor would

[18] Exton *et al.* (2010) argues that this cost furthers the digital divide mentioned above.
[19] Wolff (2006) implies that market pressure caused by the availability of localised software from volunteer and open source efforts might lead to localisation for commercial software. Honkela *et al.* (1997) note that localising for minority languages as a gesture of goodwill may affect revenue in other markets positively.

violate consistency across locales. Similar concerns apply to locale-adapted designs (Hall, 2002), navigation and layout (Abufardeh and Magel, 2010; Collins, 2002). It might be this requirement for consistency which leads to localisation often restricting itself to textual and linguistic aspects of software.

2.4 Scope of Localisation

Earlier in this chapter, the ambiguity of culture was discussed, locale was introduced as an indicator replacing culture, and localisation was defined as adaptation of software for different locales. This section will explore what adaptations can or need to be made in software. This localisation scope significantly influences both scale of software internationalisation, and localisation deliverables, and has a significant impact on the extent of developer-translator collaboration. The importance is implicitly acknowledged by software localisation models and frameworks specifically aiming to tackle culture in the context of software development, e.g. the model by Sturm (2002), or the models and processes developed by Stamey and Speights (1999) and Smith et al. (2004).

Chavan *et al.* (2009) lists language, aesthetics, religion, popular culture, history, geography and climate, among others, as objects of localisation. Tarquini *et al.* (2010) has made an effort to give a higher-level categorisation of these elements into linguistic, cultural and technical items: Linguistic items include right-to-left languages, scripts and character sets. Cultural items include legal regulations, representation such as symbols, addresses and currencies, and political and business conventions. Technical items include keyboard inputs, local service providers, layout resizing, and foreign script support.

Kumhyr *et al.* (1994) makes a conceptually different categorisation, distinguishing between product-independent adaptations required regardless of functionality, e.g. the need to translate UI text, and product-dependent adaptations specific to what software does, e.g. considerations of regional tax law in accounting software.

The locale-adaptation classifications of several authors distinguish representation and functionality. Lagus *et al.* (1997) distinguished between adaptations based on cultural factors derived from customs and beliefs and local conventions, both of which mostly affecting representation, and local practices, i.e. formulae and processes, which affect software on a functional level. A similar classification with a stronger focus on quality and

localisation depth from the somewhat different medium video games is given by Thayer and Kolko (2004), who distinguish between three levels of scope and complexity: Basic localisation translates only text. Complex localisation further adapts GUI and icons. And in blending, look and feel, functionality and usability are adapted to match requirements of a different culture.

This categorisation seems conceptually similar to the distinction of the following three levels of cultural HCI adaptation of del Galdo and Nielsen (1996):

1. The interface is able to process user's language, script, and formats, which in the opinion of the authors is achieved in most products.
2. The interface has been subjected to common usability methods in order to make it usable and understandable for international users.
3. The system accommodates cultural characteristics of the user, for example by moving the design beyond offensive and nonsensical icons to address specific cultural values such as the way communication and business is conducted.

It stands out that on the first two levels, del Galdo and Nielsen (1996) talk about capabilities of the UI, whereas they refer to system capabilities for the highest level. The five levels of localisation given by Honkela *et al.* (1997), i.e. none, minimum, moderate, high and complete localisation, do not distinguish between either representation and functionality, or different UI element classes.

There is the implication that there is localisation of presentation, and localisation of behaviour (see e.g. Abufardeh and Magel, 2008a). Schäler (2007) distinguishes between shallow level and deep level localisation: the former considers cultural conventions such as colours, symbols, sounds, signals, and product names, whereas the latter considers underlying value systems. Ito and Nakakoji (1996) distinguish between functional design so that a product is usable by foreign users, and good international HCI design.

2.4.1 Localisation Requirements

So, specifically what needs to be internationalised in software, i.e. made localisable? Obviously, UI text should be translated so that it is natural and user friendly, i.e. clear and consistent (O'Sullivan, 1989). Correct language is important as errors can lead to a decrease in acceptance. For example, simple spelling mistakes can insult Arabic users

when they are addressed in the wrong gender (Abufardeh and Magel, 2010). This is a particular concern when text is modified at run-time, e.g. if placeholders are filled in. Visually correct representation of text also relates to data representation, i.e. formatting and display, and relates to operations such as sorting and collation sequences (Kokkotos *et al.*, 1997; Law, 2003).

Beyond the obvious translation of text, localisation also needs to consider visual aspects of the UI. The locale dependence of colour associations is discussed in Barber and Badre (1998), Russo and Boor (1993) and Badre and Laskowski (2001). For example, in the Western world the colour associated with death is black, whereas in the Middle East and Southeast Asia, it is White. Symbols, icons and imagery carry different meanings in different locales. For example, the thumbs-up gesture has a positive connotation in Western culture, but is an offensive insult in Persian countries. Conversely, examples of non-Western symbols likely undecipherable for Western readers are given by Marcus (1996). Further, subtext and perceived aesthetic of sound and music vary widely between cultures (O'Keeffe, 2009).

The previously mentioned adaptations are mostly representational differences, or functionality closely associated with representation, such as list sorting. Other localisation requirements relate to software behaviour rather than presentation. Such differences are more difficult to translate into different locale-dependent requirements. A starting point are the previously mentioned operations related to data presentation, e.g. sorting, and extend to any locale-related software business rule, i.e. account and financial rules and logistic and operations practices (Abufardeh and Magel, 2010; Hall, 2000; Abdelnour-Nocera *et al.*, 2003). Considerable differences in infrastructure could lead to further shifts in software requirements. Smith *et al.* (2007) relate how a lack of fixed power lines or a reliance on alternative energies might influence engineering choices, e.g. towards batch processing. Nardi (2011) discuss cultural and physical influences on the process of withdrawing money. And the more cultures differ, the broader the adaptations that are required: Another banking-related example by Hall *et al.* (2002) describes the capability of Indian banking websites to give to charity or conduct rituals to bless a financial transaction.

Localisation requirements are regularly associated with even higher levels of cultural differences, e.g. it is often pointed out that Western and Southeast-Asian cultures are fundamentally different in problem-solving strategies, where the Western approach is based on analytic reasoning and cause-and-effect-thinking, and the East-Asian approach is holistic and dialectic (see Rauterberg, 2006; Christiansen, 2010). Zahedi *et al.* (2001) proposed cultural factors believed to determine web document effectiveness.

Many examples of cultural differences could be discussed. However, in order to arrive at a well-localised product, such punctual evidence of cultural differences have only limited usefulness as guidepost to decide comprehensively what to localise and what not to localise: First, the more subtle and subconscious the cultural differences become, the more likely it is that they will not be noticed or cannot be expressed in gritty examples as those above. Second, obvious cultural differences do not necessarily translate into what would seem to be the logical consequence in UIs, so for example the previously mentioned colour association of white with death Japan contradicts empirical data as Japanese web sites use white to a large extent (Cyr and Trevor-Smith, 2004)[20]. Third, for many cultural differences and idiosyncrasies, it is not clear how to translate them into localised software behaviour.

In practice, a more empirical approach of defining localisation requirements is applied, e.g. by having product adaptations to specific markets being guided by examining respective users in the market through ethnographic studies (Liu and Zhang, 2011).

2.4.2 Locale-specific Design and Cultural Marker

This begs the question how it is known what elements in software are locale-dependent. Is there empirical evidence?

Barber and Badre (1998)[21] identified culturally relevant conventions on websites through statistical means and usability inspections of several hundred websites from different countries and languages, and used these insights to develop guidelines to increase the cross-cultural usability, or *culturability*, of web sites. Juric *et al.* (2003) identified general issues of cross-cultural web design and found culture-specific design elements of South

[20] Similarly, Sun (2002) questions the usefulness of cultural guides advising Asian developers on the meaning of the colour red in the USA.
[21] The study is also reported in Badre (2000).

Korean and UK websites, particularly regarding colour, menu layout, and animations. Cyr and Trevor-Smith (2004) analysed 30 municipal websites each of Germany, Japan and the USA in order to find culturally preferred design elements and the degrees of difference, and confirmed variance across cultures for language and script, layout, symbols, content, structure, navigation, external links and colours.

These studies assumed that localisation or lack thereof have an effect on performance and hence usability, and concluded that usability must be defined in terms of cultural context as culture defines what is useable and what is not.

While the previous studies identified cultural markers bottom-up, i.e. based on data, a number of other studies attempted to find locale-specific software aspects through top-down methods only, i.e. by conducting content analysis and using cultural models as a framework. Marcus and Gould (2000) mapped cultural properties of Hofstede's - at the time five - cultural dimensions to implementations of websites from various locales, concluding that depending on the context, extensive adaptations of websites for different locales would be necessary. Some of their postulations were later confirmed by other researcher's results, e.g. culture-dependent website navigation was supported by Cyr (2008).

Ahmed *et al.* (2008) explored cultural values of Malaysia and Britain with relevance to web sites by conducting a content analysis for three Malaysian and British websites each. They examined how Hofstede's individualism and collectivism dimension and Hall's high vs. low context dimension was reflected in the websites and found considerable differences in cultural values on Malaysian and British websites, concluding that the world-wide web is not culturally homogeneous.

Choi *et al.* (2005) used interviews with Finnish, Korean and Japanese participants about videos showing mobile data service use to examine the relationship between four cultural dimensions derived from Hofstede and Hall, and design attributes of mobile data services for feature phones. They arrived at a list of 52 design attributes related to culture with relevance for the design of future mobile data services.

Stamey and Speights (1999) conducted a case study that combined bottom-up analysis of existing websites with top-down theories of culture to develop a methodology for localising US-American websites for Mexican customers.

Limitations of the previous studies include the predominant use of websites and restrictions to examining aspects of presentation while neglecting function or behaviour. When content analysis is not formal, there is a chance that the locale-dependence of the identified software elements and their mapping to existing cultural models is more a product of the researcher's wish, than actual locale-dependency and a data-model fit. As Sun (2002) criticised, such studies assume and confirm an incorrect view of culture as a static collection of cultural markers, although culture is dynamic and cultural markers are nothing more than its manifestation. Nonetheless, it can be expected that as computers become more ephemeral, ubiquitous and pervasive, new interfaces and modes will provide more openings for culture to play a role (Harper *et al.* (eds.), 2008). e.g. in augmented reality (Rauterberg, 2006), robotics (Levy, 2007; Weiss and Evers, 2011; Evers *et al.*, 2008), or mobile devices (Chiaro, 2009; Grigas, 2014).

2.4.3 Cultural Markers and Usability

An issue with the studies mentioned in the previous section is that in isolation, they do not actually show the necessity of cultural adaptation: Just because software products differ between two cultures does not necessarily mean that this difference has to be reflected in future products for success, particularly if the differences are primarily optical. They might also be chance or fashion, and simply irrelevant.

This would be in conflict with the assumption that the presence of cultural markers has a positive influence not only on product acceptance, but also on usability and user performance (Barber and Badre, 1998; del Galdo, 1996; Hall, 2002). For example, text input for Indian characters on mobile phones might be affected by the need for 38 key presses in order to enter certain Sanskrit characters (Clemmensen, 2010). This has been verified by so-called *cultural usability tests*.

Choong and Salvendy (1998) conducted a study showing that with regards to performance and UI, there are differences between American and Chinese users.

Americans performed better with alphanumeric and mixed icons, Chinese users performed better with pictorial and mixed icons.

Badre (2000) examined the relationship between culture and website design and usability, showing that websites with Italian cultural markets required fewer clicks for navigation on average than with US cultural markers, and that US subjects might prefer foreign designs, but do not perform better on domestic websites.

Aryana and Liem (2011) examined usability differences for Turkish and Iranian mobile phone users through interviews, focus groups and usability studies. Cyr (2008) found that localising websites increases trust, satisfaction and customer retention, i.e. repeat visits, in e-commerce websites.

The results align with theoretical predictions. The UI is one of the most important aspects of software, and its quality is determined by UI text language and design (Irmler and Hartwig, 2000). Although users learn quickly to link a representation, e.g. a symbol, to its underlying function, understanding a representation facilitates this learning (Liem *et al.*, 2011), and cultural background has a strong influence of the understanding of UI elements (Smith *et al.*, 2007).

Sun (2004a, 2004b) examined users of text messaging in the US and China through surveys, observations, interviews and diary studies, and concluded that localisation of operational functions ignores that international users then effectively use the localised product differently, with different goals and objectives, than users in the original locale. The author suggests to localise for concrete use cases within specified contexts and to consider social aspects during localisation.

However, although usability and related factors such as dependence and acceptance are important factors for the success of software (Sommerville and Dewsbury, 2007) and individuals are more likely to interact with technology if it appears easy to use and appealing (Agarwal and Karahanna, 2000), the research results discussed above are limited to a subset of global locales. Further, most studies on culture and HCI are questionnaire-based quantitative cross-cultural comparisons conducted with university students who more often than not speak English, and are analysed on the level of

national groups, not regions (Clemmensen and Roese, 2010). Hence, there is only limited validity and applicability of the results.

There are also indications that for users, locale-dependent performance is not paramount. First, a lack of cultural fit can be compensated by users (Sun, 2004b). Second, some users appear to consider unlocalised software good enough or even superior (Cyr and Trevor-Smith, 2004; Hall, 2006). Third, socio-economic rationales and preferences can supersede individual preferences (Wolff, 2006; Schäler, 2007; Hall *et al.*, 2009).

2.5 Localisation Factors, Issues and Challenges

A localised product must meet local needs while reaching the market within reasonable time (Hudson, 1997). Additionally, as for any project, there is also a limit on cost that can be accrued and of effort that can be expended. The *localisation factors* cost, quality and time (Hudson, 1997) are the criteria of successful software localisation (Karkaletsis *et al.*, 1995).

Localisation costs can be considerable (Collins, 2001; Ryan *et al.*, 2009). For example, DePalma (2006) report a survey where 0.25% and 2.5% of the international revenue was spent on localisation, but noting that exact accounting of localisation cost to locales, products and activities is difficult. External or exclusive costs such as staff costs for translators are comparatively simple to break out, but internal localisation costs, i.e. efforts of project managers and software engineers for which internationalisation and localisation work and localisation bug fixing is only part of their work, is complex (DePalma, 2006). Further costly localisation expenses include software purchases, compilation of information kits for localisers, localisation testing and terminology setup (Honkela *et al.*, 1997). Localisation cost is also linked to volume, i.e. word count and number of target locales (Ryan *et al.*, 2009), as a localisation service provider (LSP[22]) or freelance translator charges per word and each target locale requires at least one additional translator. Another major contributor to localisation cost is the handling of localisation bugs (O'Sullivan, 2001a). Further costs might be created through the purchase of localisation tools. As with many other efforts, although localisation cost cannot be known definitely while product development is ongoing, they must be forecast

[22] Also called localisation vendor or translation agency, though the latter technically offer different services.

nonetheless (Sikes, 2011). A noted imprecision of the term cost is its implication of an exact, clearly defined financial value such as the salary of translators. However, many localisation-related costs do not appear separately on spreadsheets and are difficult to determine, i.e. the time spent by software engineers satisfying internationalisation requirements in the software or fixing localisation-related bugs.

Localisation quality has not been standardised yet (Lewis *et al.*, 2009; Tarquini *et al.*, 2010). McHugh *et al.* (1997) suggest to understand localisation quality in terms of the ISO 9126 standard[23] (ISO/IEC JTC 1/SC 7, 1991), which defines software quality through functionality, reliability, usability, efficiency, maintainability, and portability. These would accordingly be mapped to meeting local user needs, apparent friendliness towards a local user, software effectiveness in the locale, documentation correctness, difficulty of internationalisation and localisation, and extent of internationalisation and localisation. Alternatively, McHugh *et al.* (1997) suggest to understand localisation in terms of technical quality, linguistic quality, and how a product compares to the competition.

Localising a product takes time (Collins, 2001), with the main concern being the impact internationalisation and localisation have on the release date (Caesar and Fehrenbach, 2005). From the software industry's point of view, time to market is extremely important. Due to the high frequency of innovation, being second risks losing the market, so even moderate delays can have disastrous effects on success. Accordingly, the software industry has developed an almost obsessive relationship with time to market (Boehm, 2011, 2006; Blackburn *et al.*, 1996).

Software localisation has been impacted by this through the practice of *simshipping*[24], i.e. the simultaneous release of all locale versions of a product (Ryan *et al.*, 2009; Hartley, 2009; Zhou, 2011), a requirement often originating in marketing departments (Kahler, 2000). Even a moderate time between domestic and localised availability allows competitors to develop and release a competing, localised product and seize the market of a locale. Further, many customers do not want to wait for localisation and instead obtain a domestic version. This might impact the acceptance of localised versions and

[23] ISO 9126 has since been superseded repeatedly. The current standard for software quality is ISO 25010 (ISO/IEC JTC 1/SC 7, 2011).
[24] The opposite of simshipping, i.e. localisation after the release of the initial version, is called *post-release* (Zhou, 2011).

drive software piracy when a domestic version is not legally obtainable. In extreme cases, there is a danger to misinterpret any delay between releases of different locale versions as slight or preference of rival locales, leading to reputation loss for company and product. For example, Edwards (2012) warns that delays between the release of Hebrew and Arabic versions might lead to such sentiments in the locale for which the product was released later.

Simshipping is believed to increase pressure on localisers (Ryan *et al.*, 2009). Estimates vary wildly, but it can be assumed that translators can translate between 1,000 and 2,000 words per day (see Combe, 2011). Because UI texts can easily contain more than 10,000 words, delaying a product release until translation is finished can cause significant delays. On the other hand this parallel working of engineering and localisation is credited with benefitting complex localisation projects where application functionality has to be localised and engineering can react to input from localisation (Zhou, 2011). However, simshipping and parallel engineering and localisation has also been associated with increasing the difficulty of estimating cost and time in project management (Sikes, 2011).

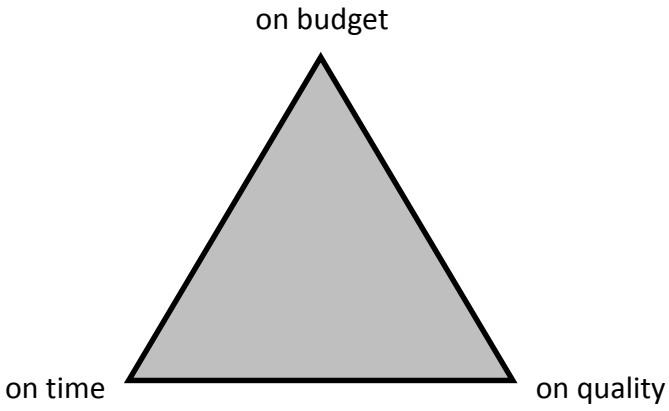

Figure 2-1 The project management triangle

The three factors cost, quality and time map to the so-called project-management triangle. The idea behind it is that a project is located somewhere in the triangle spanned by the three factors, and can move towards any one factor, or within limits decrease the distance to two, but never towards all three. In fact, getting closer to one factor always comes at the cost of increasing the distance towards at least one of the other two factors. The applicability of this concept to localisation can be illustrated by Collins' (2001) discussion of on-site localisation: Having translators on-site increases quality, but comes

at a higher cost than off-site localisation. Dunne (2011) applied the project management triangle to localisation, noting that in localisation projects, "[c]ost rules, quality is assumed, but in the end, schedule wins" (Dunne, 2011, p.120).

2.5.1 Localisation Issues

In the previous subsection, three localisation factors were introduced: cost, quality and time: Regular software localisation and internationalisation incurs cost and takes time, and delivers a product with a specific localisation quality. This relationship was already elaborated on in the previous subsection. This subsection will look at localisation issues, i.e. problems related to localisation. These can be separated into product-related and process-related problems.

The most obvious localisation issues are probably product-related problems, i.e. quality problems with the software, or in other words, localisation-related bugs. A localisation bug is anything in the final software product that can be classified as an error. Localisation bugs range on a severity scale from comparatively minor, such as a disputable translation leading to a minor aesthetics issue, to serious, such as a malformed placeholder leading to abnormal program termination. In between are incorrect, ambiguous, partially displayed and missing localisations and translations, incorrect vocabulary, disregarded writing conventions such as direction, punctuation, spacing rules, sorting and collation, or failure to consider locale-dependent techno-cultural aspects such as character encoding, keyboard shortcuts and technical infrastructure (Ryan *et al.*, 2009; O'Sullivan and Hyland, 2004; Collins, 2001; Pérez-Quiñones *et al.*, 2005; O'Sullivan, 1989).

Localisation quality has been the subject of various studies examining translation quality, consideration of locale requirements, and applicability of the use case for the international market (O'Sullivan *et al.*, 2003). Localisation bugs can lead to usability issues, and the link between usability and localisation has been discussed earlier. Even if no usability issue is apparent, a – possibly subconscious – effect of localisation bugs on product acceptance should not be underestimated as the effect of localisation bugs can range from inconsequential annoyance to perceived offense. For example, Abufardeh and Magel (2010) explains how simple writing mistakes might insult Arabic users who are addressed in the wrong gender. DePalma (2006) elaborates that depending on the market, perfection for its own sake can determine product success.

However, localisation issues can go beyond the product itself and affect the process, e.g. in the form of superfluous engineering and localisation efforts incurring unnecessary cost and causing unnecessary delay of the final software. Such issues include duplication of effort (Hogan *et al.*, 2004), particularly repeated translation of text for which translations already exist from previous translations. Additional cost and delay can be caused by quality assurance and quality control efforts as a response to localisation bugs. In fact, the impact the correction of localisation bugs has on product cost and schedule at one time was believed to be so huge that O'Sullivan (2001a) stated that complexity and time for fixing localisation bugs is the main cause of localisation costs. However, as discussed earlier, as these would be internal localisation costs, it is difficult to separate them from other internal costs unrelated to localisation[25].

2.5.2 Role Relationships and Causes of Localisation Issues

Localisation issues are not of equal concern for each role involved in developing international software. Some errors concern engineers, others concern translators or linguists, even others concern project managers, and so on (O'Sullivan and Hyland, 2004). For example, issues-related localisation concerns from an engineering perspective might be whether the application has been properly internationalised, i.e. all locale-relevant content has been completely separated from the code, whether all content storage files adhere to standardised formats, whether all UI text can display any possible Unicode characters, whether all functions processing strings can handle non-Latin languages and script, and so on (O'Sullivan and Hyland, 2004).

Equally, different sources of localisation issues have been identified in the literature. Many localisation bugs are caused by incomplete or incorrect internationalisation (Pérez-Quiñones *et al.*, 2005; Ryan *et al.*, 2009; Hogan *et al.*, 2004; O'Sullivan, 1989). These, and localisation bugs caused by lack of contingency for text expansion during translation, have been attributed to lack of knowledge of software engineers (O'Sullivan, 2001a; Pérez-Quiñones *et al.*, 2005). The consequence of lack of knowledge about culture which leads to ethnocentric misconceptions in the design of software products in the form that it is

[25] The "thousands" of localisation bugs observed by O'Sullivan (2001a, p.7) will probably not occur in software developed with today's technology since arguably many of those bugs are avoided by the use of localisation APIs and localisation frameworks. Nonetheless, the link between localisation quality issues and cost and time remains.

assumed that interpretations and understanding of software functionality are universal across different cultures, when they are not[26] (Vatrapu, 2011). Accordingly, such localisation bugs are not caused by localisation, but are an inherent property of the English product (O'Sullivan, 1989). Besides quality, Law (2003) reported an impact on cost and time through the need for additional quality assurance and quality control caused by lack of understanding of culture.

A lack of planning, communication and coordination have also been identified as source of localisation issues:

Good translations require communication between translation and engineering roles to remove ambiguities of the meaning of the source text (Russo and Boor, 1993; O'Sullivan, 1989). Usually, this requires the translator to know the context of the text to be translated. Context is information informing about the situation of an item, e.g. person, place or other object, relevant to the interaction between user and application, including user and application themselves (Aryana and Liem, 2011). For example, the term *manual download* might refer either to the download of an instruction manual, or a manually initiated download as opposed to an automatically initiated download. Considering that not all target languages can cover both meanings with one term, a translator has a 50% chance of picking the wrong meaning with merely the text to go on, and thus providing an incorrect translation (Freigang, 2000).

The responsibility for the lack of communication has particularly been assigned to localisation agencies, and when designers and programmers were uncooperative towards translators with respect to providing context information (O'Sullivan, 1989; Combe, 2011; DePalma, 2006; Honkela *et al.*, 1997).

It is often stated that this kind of communication is handled by so-called *localisation kits* including material giving the translators context for the source content (Honkela *et al.*, 1997). However, employing such unidirectional communication through localisation kits relies on the assumption that there are either no issues for translators to note, or no questions for them to have. Neither assumption is correct (Sikes, 2011).

[26] Some examples of Western products failing in international markets are given in Chavan *et al.* (2009).

2.5.3 Future Localisation Challenges

In the development of global application, previous challenges had been identified as the adoption of Unicode, and advanced internationalisation architectures (Law, 2003). In the literature, the expectation has been voiced that there will be continued pressure to reduce localisation cost and time, increasing localisation quality, and that localisation volume will continue to increase (Ryan *et al.*, 2009).

The use of technology, for example of machine translation and computer-assisted translation, will equally continue. Increased use of such technology will continue to enable the spreading of translation jobs over several translators in parallel. Together with translation from updated material, translators can expect to translate more and more fragmented source texts (Bikmatov *et al.*, 2013; Esselink, 2003).

This development also robs translators of context. Accordingly, context provision to translators and enabling translators to preview their translations in the eventual publication format, e.g. the final UI, have been identified as another challenge to overcome (Bikmatov *et al.*, 2013).

2.6 Facilitation and Support of Localisation

In the previous sections motivations, scope of localisation, issues, and perceived future challenges were discussed. This section will look at practice, existing research and development towards decreased cost and time, and increased quality.

Because the building blocks of computer programs, e.g. source code and UI definition, are generally generated on a computer, it is only natural to localise software, i.e. adapt or translate said building blocks, using a computer as well. Accordingly, software has become a tool to facilitate and improve localisation. Factors facilitating localisation have been identified as translation tools, platform support, Unicode and UI guidelines (del Galdo, 1996; Vouros *et al.*, 1997). Additionally, this section will discuss outsourcing and standards.

2.6.1 Translation Tools

Translation tools refers to software helping a translator to translate. So-called Computer-Assisted Translation (CAT)[27] tools can be categorised into three types of *linguistic resources* (Lenker *et al.*, 2011): translation memories, machine translation, and terminology databases. The lines between them are blurring (Reineke, 2005), and these linguistic resources are often combined into one application called a *translator's workbench* or *translation editor* and include text processing functions such as spell, syntax and style checking as well as import and export functions for various file formats, including source code and binary files (Vouros *et al.*, 1997; Freigang and Reinke, 2005). Since these tools improve the task without changing it radically, Exton *et al.* (2010) classified them as *enabling technologies*. A case study of CAT tools was conducted by Schäler (1994), finding an increase in translation speed, improved consistency in translations and therefore improved translation quality, and decreased translation cost. Wolff (2006) noted that certain features of translation tools, e.g. automatic spell checking and automatic checks for correct punctuation, spacing, and placeholder use, enable non-native speakers of a language to at least determine what translations require a review, thus lowering the workload of translators.

On the other hand, usage of translation tools comes with its own issues and disadvantages. Translation tools require considerable training (Bowker, 2005; Wolff, 2006; Moorkens, 2012a) but affect work practices and are therefore controversially viewed by translators (Wolff, 2006; Stoeller, 2011). Translation tools have also been found to impose the tool developers' interpretation of translation and localisation onto translators (Hartley, 2009; Dohler, 1997), introduce new problems into work processes that are difficult to mitigate (Schäler, 1994), and the maintenance efforts required to ensure their effectiveness is often underestimated (Sikes, 2011).

Nonetheless, it is expected that the use of translation tools will increase (Yuste, 2005), as will their functionality and their adaptation to the needs of translators and localisation teams (Irmler and Hartwig, 2000).

[27] CAT tools are those tools directly related to translation and localisation. General-use software with merely the potential to assist in localisation as a general industrial process, e.g. workflow and content management tools, are not discussed here.

2.6.1.1 Translation Memories

A Translation Memory (TM), also called Translation Memory System (TMS) or repetition manager, is a software tool that keeps track of previously used translations. TMs store source text and its corresponding *aligned*[28] translations in the form of *translation units* into which the text has been *segmented*, usually along sentences, headings etc. (Bowker, 2005). The fundamental idea behind TM is that every translation is available for subsequent use. This could either be continued use, for example if an already translated string is moved to a different file or place, or re-use, for example if a new string has previously been translated in a different place or project. Existing translations of identical source texts are called *100% matches*, and those of similar source texts *fuzzy matches* (Freigang and Reinke, 2005; Bowker, 2005), with a percentage value expressing how similar the matches are.

TM usage comes with the side effect of increasing effectiveness of collaboration between translators. It is increasingly common to have multiple translators translate a body of text (Munday, 2009), which is perceived to threaten consistency as translations by different translators diverge (Vouros *et al.*, 1997; Law, 2003). The ability of TMs to compare translations of similar units is supposed to help translators keeping their translations consistent.

Obviously, TMs work best for translation of text with a certain amount of repetition and have been identified as tools specifically for technical translation (Hartley, 2009). Their use is practically indispensable in localisation for its effected translation speed increase (Yuste, 2004; Lenker *et al.*, 2011; Moorkens, 2011), confirmed through case studies (e.g. Schäler, 2007; Bauer and Rodrigo, 2004) and experiments (Bowker, 2005). Bowker (2005) also gives comprehensive insight into how TMs are used in practice and provides numbers: Efficiency gains through TM usage can range from 10% to 70%, with 30% being considered a realistic figure.

Obviously, however, the translation quality within a TM is always dependent on the original translator, and TM usage makes most sense when working with repetitive, frequently revised or updated text. Further, the efficiency increase requires the overall

[28] Alignment is usually done during translation. If done retroactively for existing corpora, this is called *post-translation alignment* (Bowker, 2005).

context of translations not to deviate much (Bowker, 2005; Moorkens, 2012a, 2012b). It is hence recommended to exclude text with uncommon terminology or unique style from TMs (Bowker, 2005) and be careful when mixing text from different departments or times in TMs. TM efficiency also increases when applying source text standardisation such as controlled language (Hudson, 1997; Allen, 1999; Moorkens, 2012a), i.e. language restricted in grammar, vocabulary and syntax in order to lower complexity, avoid ambiguity and provide consistency (Vouros *et al.*, 1997).

While the advantages of TM usage are acknowledged, research results also indicate a number of limitations and disadvantages: The more analytic a language is, i.e. the less the grammar requires words to be modified, the more helpful TMs can be, whereas the more synthetic a language is, the more difficult it is to apply such tools. Respective difficulties for Baltic languages are discussed by Rusakevičienė and Kriaučionytė (2012). Further, Schäler (1994) noted that the tool enforced an inflexible work process on users which is likely to slow down experienced translators, and Bowker (2005) found that while use of a TM increases translation speed, quality dropped at the same time as translators are tempted to be uncritical about translation proposals coming from a TM. Moorkens (2011, 2012a) identified, categorised and measured consistency in TMs, finding as considerable cause clients' focus on time and cost savings over quality and a wide range of clients' often incorrect assumptions regarding translation practice. These observations confirm that TM usage removes context and increases distance between translator and source text, requiring additional pre- and post-translation work (Ottmann, 2005).

Bowker's study results are discussed by Pym (2008) as manifestation of Toury's laws of growing standardisation and of interference, stating that features of the source text, such as metaphors, become a regular target language feature. The law of interference predicts the carrying-over of source text characteristics, such as structure, into the translated text. In short, Pym concludes that usage of TMs increases standardisation in translations, but also properties which are uncharacteristic of the target language. While the former is not all bad since it leads to increased consistency in translations, that is, precisely the benefit TMs are supposed to produce, Pym criticises the practice of segmentation as such because imposing the source text's segmentation makes a translation more difficult to comprehend to the point where limitations of TMs have more influence on the

translation than the actual source text and there would be no net improvement in comprehensibility through the use of TMs.

Bowker (2005) and Moorkens (2012a) also points out that identical source text segments can warrant different translations based on the context, i.e. the preceding or succeeding segments in the text. Consequently, identical source text can and should not necessarily be translated consistently.

2.6.1.2 Machine Translation

Machine Translation (MT) is the automated translation of text through software. There are two main families of MT methods: rule-based machine translation, also known as classical approach or knowledge-based machine translation, based on codifying a language's rules into software, and statistical machine translation, inferring translations by statistically analysing a corpus of available parallel texts of source and target language.

MT currently still has weaknesses considering the context for translations, e.g. non-literal meanings such as irony, ambiguity or humour (Morado Vásquez *et al.*, 2011). In software localisation, concerns regarding MT revolve around the field's typical rapid invention and change of novel technical terms for which no equivalents exist yet in target languages, further the abbreviated UI language with little grammatical structure, single-word labels, in particular homographs, and the mix of technical instructions as part of text, e.g. placeholders and inclusion of keyboard shortcuts (Hogan *et al.*, 2004; Elsen, 2005; Kumhyr *et al.*, 1994).

Although there is no agreed measurement of translation quality (Lewis *et al.*, 2009), various methods of assessing MT quality and MT strengths and weaknesses (e.g. Bohan *et al.*, 2000; Vasiļjevs and Sāmīte, 2012) have been examined. Yao *et al.* (2002) computationally measured the quality of software localization-oriented MTs by running a number of string comparison algorithms against machine and human translations and measuring the string differences, statistically showing computational results to correlate with human evaluations. Pérez-Quiñones *et al.* (2005) examined the quality of a commercially available MT software through comparison with crowdsourced and human translations, and the suitability of back-translations for the evaluation of MT quality.

Nonetheless, there is a trend towards more translation automation (He *et al.*, 2002), with MT eventually taking over the task of translating altogether. Enabling good translation now might lead to a solid basis for MT in future projects (Irmler and Hartwig, 2000). However, even in that case, translators will shift their competences towards terminology work, TM maintenance, terminology work[29], corpus maintenance, application of controlled language, or simply switch to more sophisticated translation tasks not suitable to MT and projects requiring careful assessment of cultural issues beyond translation (Yuste, 2005). In fact, this practice has already been reported from the industry, e.g. by Hudson *et al.* (1997).

MT has already found its niche applications in software localisation and globalisation strategies, e.g. in online communication, social networking and other applications of what Elsen (2005) calls *gisting*, where quick or cheap availability of approximate meaning is more important than a perfect translation (Morado Vásquez *et al.*, 2011). Obviously, these are dramatic differences in translation requirements (Lewis *et al.*, 2009; Morado Vásquez *et al.*, 2011), and Bauer and Rodrigo (2004) differentiates such *receiver-commissioned translation* from the classic *sender-commissioned translation* where quality demands remain high. Examples of respective applications of MT are described in Thicke (2012), Stewart *et al.* (2010), and Porsiel (2008).

2.6.1.3 Terminology Databases

Terminology refers to a systematic collection of words and terms and their meanings in defined contexts. A terminology database is similar to a glossary, but goes further, aiming to increase consistency in source texts and translations by providing a corpus of terms and definitions of their meanings for use by source text authors and translators (Bowker, 2005; Vouros *et al.*, 1997). The need for terminology databases has been identified in localisation case studies, e.g. in Law (2003).

Active terminology lookup during translation is supposed to increase translation quality (Bowker, 2005), and managing terminology in a source text is assumed to increase the efficiency of MT and TM for during translation dramatically (Bowker, 2005; Moorkens, 2012a). However terminology databases require a significant amount of setup work and

[29] Terminology management is discussed in subsection 2.6.1.3.

ongoing and maintenance effort by dedicated staff, translators and content creators (Karkaletsis *et al.*, 1995; Bowker, 2005; Bauer and Rodrigo, 2004; Schubert, 2009; Vouros *et al.*, 1997). Terminology should be translated before the actual content (Vouros *et al.*, 1997). The incorporation of a terminology database into product development and localisation is illustrated in a case study by Bauer and Rodrigo (2004).

2.6.1.4 Provisioning Locale-specific Information

The examination of cultural differences and the scope of internationalisation and localisation has been discussed in subsection 2.4. Of course, identifying cultural markers and applicable cultural models is only half the work. At worst, it leaves a lot of doubts what really needs to be internationalised, and at best, it says what needs to be internationalised, but not how.

A number of authors suggest development some kind of repository containing data or information which helps directly or indirectly with localisation. The idea of storing cultural information for various purposes is not new. For example, Bumeder *et al.* (2003) presented a repository for improving intercultural collaboration. With respect to software localisation, Ryan *et al.* (2009) focuses on reducing localization costs by a tool called Localisation Knowledge Repository (LKR), a library containing development guidelines regarding content, presentation, navigation, accessibility, and other issues compiled from primary research, secondary research, existing literature and best practice. The intention is for LKR to incorporate internationalisation guidelines into the development process (Anastasiou, 2009).

Assumedly, results from culture-HCI studies would feed into such a database as suggested by Ryan *et al.* (2009). Similar ideas also appear in publications of other researchers, i.e. a central repository for culture-specific information (Mahemoff and Johnston, 1998) and "resource banks of local knowledge so that developers can avoid misunderstandings" (Smith and Dunckley, 2007, p.2).

Hall *et al.* (2003) argue that guidelines across cultures for the development of UI interfaces are not optimal because they are specific to the culture in which they were developed. Instead, the authors suggest to use design patterns as an aid to design cultural UI because these "encapsulate context" (Hall *et al.*, 2003, p.87). They introduce design

patterns as an alternative to guidelines, as a solution to a problem subject to a specific situation or context. Patterns all contain a statement of the problem, the context in which the problem occurs, and a description of one or more proven solutions in that context. The authors emphasize the key difference between guidelines, which are rules-driven, and patterns, which are data-driven. Guidelines are hence culturally specific, fail to consider context, are difficult to apply to specific cases and have no internal structure. To a given problem, there should be a different pattern for each culture. The authors link this to Trompenaars and Hampden-Turner (1998) explaining culture as the problem-solving strategies of groups of people, with each pattern having been proven to be effective in providing a solution for each culture. The authors further suggest a development of "pattern calculus" (Hall *et al.*, 2003, p.90) so that unknown design patterns for cultures can be calculated from existing design patterns from similar cultures.

However, the idea of cultural design rules has critics: Collins (2002) points out a drawback of design rules, suggesting that any effort to come up with design rules for localization will eventually be stereotypical and miss cultural variations.

Liem *et al.* (2011) points out that the established design standards, rules and guidelines usually fail to address issues relating to culture for two reasons: First, the use of standards, rules and guidelines does not guarantee good design to begin with - standards, rules and guidelines can just as well justify bad design. And second, cultures are inherently subjective and hence cannot be objectively described – they are not ontologically objective. Consequently, cultures cannot be objectively described, measured, or codified into guidelines. Similarly, Sun (2002) interprets cultural guidelines as a manifestation of a positivistic scientific view that is not applicable to culture. Kamppuri (2011, p.24) notes that guidelines and checklists for design are a symptom of the technical approach to culture in software development.

Without intending to criticise the notion of databases, guidelines, repositories etc., I note that all these papers effectively propose a kind of tool with an objective. All these tool proposals imply that the provisioning of cultural information, or of processed cultural information in the case of the design patterns proposal of Hall *et al.* (2003), will improve software localisation in some way.

However, the integration of these tools, as well as existing tools such as MT, into the software development process is not discussed. That is understandable as each software development project is different, and the applied method or process in each project differs in some way from others as well. In other words, there is no standardised software development process, and diversity among software development processes is high, as noted by various scholars, e.g. Lindvall and Rus (2000).

2.6.2 Platform Support

Platform support refers to existing code that can be exploited by software developers so that they do not have to implement respective locale-dependent code themselves. Generally speaking, platform support comes on two levels, on the operating system level and on the library level.

Operating system support handles locale characteristics and user preferences, meaning Unicode and locale support (Hall, 2000), i.e. the operating system contains scripts and a database of locales with various characteristics and provides the respective user settings and preferences to an application. Library support generally covers locale-dependent input and output handling as well as provision and handling of locale-neutral data, e.g. through predefined data types for locales, date, time, currency or Unicode strings. It usually comes in the form of an application programming interface (API) and as part of UI libraries.

Requirements and a possible implementation for such API libraries are outlined in Lehtola *et al.* (1997). Many common modern software development frameworks include internationalisation support in the form of UI abstraction and resource structures for easy externalisation of text and images. This simplifies localisation insofar as it decouples software compilation and localisation and allows for easy provision of locale-related content. Further, international software support is provided through locale support and locale-sensitive UI elements whose locale-dependent functionality is handled internally, reducing the need to write locale-dependent code for locale-dependent representation, i.e. numerical representation, sort orders, collation and so on (O'Sullivan, 2001a; O'Sullivan *et al.*, 2003; Hogan *et al.*, 2004; Hall, 2002).

In practice, software developers can create an internationalised application by using existing UI elements provided through existing APIs and relying on the API for locale-specific implementations. In the application, the intended locale is set through calling an API function at runtime, setting an application-global variable. The application can then use API function calls to instantiate the UI elements, e.g. an item list, and fill it with content, e.g. a label and items to be listed. Since the actual UI element, here the list item, was implemented by the external party to exhibit different behaviours according to the globally set locale, the software developer does not have to implement locale-specific behaviour, e.g. locale-dependent sorting rules for lists. What the software developer does have to provide, however, is localised content of the list. Modern APIs provide a framework for managing such locale-specific content in separate resource files.

A particular example of a localisation-specific API is the Babel Software Micro-Crowdsourcing architecture (Exton *et al.*, 2010). Crowdsourcing is the idea of soliciting work, often repetitive or comparatively trivial, from a large crowd, often in the form of an online community. It is already being applied practically, often to provide or proofread translations (e.g. Facebook, 2008), but requires an active community of a certain size, which might be an issue for languages, locales and software with a small population or user base (Morado Vásquez *et al.*, 2011). The Babel Software Micro-Crowdsourcing architecture combines translation-related UI editing ability with crowdsourcing management. The text of each UI element can be edited in situ without exiting the target application. UI text edits are sent to a remote server for management and coordination, e.g. through manual review.

2.6.3 Outsourcing

A very common practice in software localisation is outsourcing of text translation and other adaptation activities (Dohler, 1997; Immonen and Sajaniemi, 2003a; Yuste, 2004). Localisation is predestined for this because localisation projects require extensive resources, but only last a limited time (Combe, 2011). Giammarresi (2011) report that 87% of all companies outsource[30] their translation and localisation work. When

[30] Strictly speaking, outsourcing refers to transferring activities which previously were conducted in-house to external companies. However, it appears that a major part of the literature uses the term in a more relaxed fashion as reference to simply contracting out activities, regardless of whether they were originally done in-house or not.

translation is not generally part of their core competencies, employment of translators only makes business sense for large companies that create enough content to continually require translations and have the human resources to manage translators (Combe, 2011).

Localisation outsourcing works best in a constantly managed, long-term relationships with a vendor possessing an appropriate skill level (Papaioannou, 2005). Localisation outsourcing can increase efficiency including lower localisation cost and duration, bring flexibility in processes and organisational structures, and lower localisation cost accounting complexity. However, it also means a loss of control over processes and quality, an eventual dependency on vendors due to loss of localisation skills, and the risk of increased localisation cost and time through management overheads, communication with the vendor, and vendor profit margin (Papaioannou, 2005; Honkela *et al.*, 1997; Collins, 2001).

Because the software company and the language provider are in a buyer-seller or client-vendor relationship (Combe, 2011; Milder, 2000), localisation outsourcing is often not perceived as the long-term commitment or strategy it should be, and the need to involve the vendor in the process of creating an international product is often not seen. Giammarresi (2011) blames this on the companies' desire to shed a risk and an activity rather than purchasing a service, and laments that vendors are involved too late in the process to provide any assistance beyond simple translation. Accordingly, DePalma (2006) found that in practice, many software companies give their vendors relatively little direction and information through style guides, terminology and context information, impacting localisation quality.

2.6.4 Standards

A number of industry standards relevant to software localisation, internationalisation and translation exist. Some have been defined by the International Organization for Standardization ISO. For example, the previously mentioned ISO 639 is a collection of five standards for the naming and representation of languages and language groups (ISO TC 37/SC 2, 2002), ISO 3166 does the same for countries and their subdivisions as well as other areas of geographic interest (ISO TC 46, 2013). ISO/IEC TR 11017 defines an internationalisation framework, i.e. locale-dependent functionality to be provided by applications (ISO/IEC JTC 1/SC 22, 1998). ISO/IEC 14651 defines a method for sorting and

ordering text data and a template to define locale-relevant ordering and sorting changes (ISO/IEC JTC 1/SC 2, 2011). ISO/IEC TR 30112 defines formats and functionality for the description of cultural conventions and character names (ISO/IEC JTC 1/SC 35, 2014). RFC 5646 by the Internet Engineering Task Force (IETF) defines a language tag format for use in internet standards, protocols and documents. Trans-WS is a standard for the provision of translation web services, specifying remote function calls for the submission and retrieval of source and translated files or localised content.

The relevance of understanding standards in this research is in how they shape and influence the work of developers, e.g. by simplifying and unifying development work in the case of Unicode, and localisers, e.g. by providing a localisation process infrastructure in the form of file formats. Following, these standards are discussed.

2.6.4.1 Unicode

Unicode was developed in order to provide support for international scripts beyond those contained in single byte character sets such as the US-ASCII[31] character encoding scheme and CP-1252. These encoding schemes used 7 or 8 bits to encode characters, which means that they can enumerate 127 or 255 characters, which is sufficient for most of the characters of the Latin alphabet, but not enough to include other scripts such as the Greek alphabet, Cyrillic script, and Japanese Kana and Kanji. To display these scripts, applications have to switch encoding schemes or *code pages*, as they are called in this context. For example, the code page defined by ISO 8859-1 (ISO/IEC JTC 1/SC 2, 1998) contains all characters for USA and Western European languages, ISO 8859-2 (ISO/IEC JTC 1/SC 2, 1999a) contains all characters for Eastern European languages, and ISO 8859-3 (ISO/IEC JTC 1/SC 2, 1999b) contains the Cyrillic character set. Scripts containing more than 255 characters are encoded in so-called double-byte code pages, e.g. Japanese in CP-932 and Traditional Chinese in CP-936 (Dr. International, 2003). But code pages come with limitations such as increased software complexity (Hall, 1998).

The stated goal of Unicode is to eventually provide support for any script in existence without code pages. It is developed in tandem with the standard ISO/IEC 10646 (ISO/IEC JTC 1/SC 2, 2014), which defines the Universal Character Set (UCS). Unicode contains the

[31] ASCII stands for American Standard Code for Information Interchange.

UCS, but additionally contains specifications such as collation, sorting, and bidirectional writing, as well as additional properties for each character required by these specifications. Characters defined by Unicode can be used through a collection of so-called Unicode transformation format (UTF) encoding schemes, e.g. variable-length UTF-8 and UTF-16 or fixed-length UTF-32. All encodings are able to refer to the entirety of Unicode code points.

Ideally, Unicode is supported on the operating system level of a system (Portanieri and Amara, 1996). It is supported by all major platforms and in most modern programming languages. Adapting existing code-page-based source code to Unicode can be a complex and intensive effort depending on the circumstances such as programming language and extent of existing Unicode support. An automated conversion technique is demonstrated by Peng *et al.* (2009).

Prior to the widespread adoption of Unicode, considering code page and script handling was an essential consideration in internationalisation (Hall, 1998; Arthur, 1998). Unicode simplified internationalisation insofar as software developers do not have to worry about code pages and memory footprint limitations of different scripts and can assume that all major writing systems are supported (Hall, 2002; Law, 2003). Encoding of different writing systems into Unicode is ongoing, with Unicode 8.0 being the current version at the time of writing (Allen *et al.*, 2015). Procedural difficulties in encoding scripts arising from the interdisciplinary nature of encoding, in particular from conflicts between socio-political and technological views, are explored in Hall (1998, 2015) and Hall *et al.* (2014).

2.6.4.2 File Standards

Internationalisation means that locale-dependent information is kept separate from the program code. Often, this information is stored in separate files for each locale. For this task, a number of proprietary resource file formats have been developed over time. For example, Sachse (2005) explores resource file formats for the Microsoft Foundation Classes (MFC) and Windows resources, Microsoft .Net and for the Delphi programming platform, but also localisation-related open file formats such as the Gettext Portable Object Format, Extensible Markup Language (XML) and the XML Localisation Interchange File Format (XLIFF). Extensive research has been conducted in the development and improvement of these and similar file format standards. Many of these standards are

based on XML and were created on initiatives of the Globalization and Localization Association (GALA) (GALA, 2015) and the Localization Industry Standards Association (LISA)[32].

There are three rationales for developing open standards: First, to provide file formats suited for tasks for which no standards exist yet. Second, to improve on drawbacks and failures of existing file formats. And third, to allow exchange of localisation-related information across organisations and tools regardless of proprietary vendor-controlled formats, i.e. to prevent vendor lock-in (Anastasiou, 2009; Anastasiou and Morado Vázquez, 2010; Anastasiou, 2010a).

The majority of standard formats discussed here aim to exchange information and are based on the XML, i.e. human-readable text files using so-called tags to define content and its properties. Many of these formats include metadata, i.e. information related to the content such as author name, creation date and subject matter, which is believed to help with translation by providing context to the translator (Esselink, 2003).

The most prominent interchange format is probably XLIFF, developed to enhance interoperability and data exchange between localisation tools (Wasala *et al.*, 2012; Morado Vázquez and Mooney, 2010). XLIFF can store text and binary resources, alternative translations, and metadata. Development is ongoing, the current version at the time of writing is XLIFF 2.0 (Amaya *et al.*, 2014). Although the adoption of XLIFF has been hampered by a lack of awareness, incomplete adoption and the format's limitations, it has already achieved widespread adoption (Wasala *et al.*, 2012; Anastasiou, 2010b; Lewis *et al.*, 2009).

Another widely adopted file standard is Translation Memory eXchange (TMX) for the exchange of whole TM databases (Lewis *et al.*, 2009), although differing implementations have led to compatibility issues between tools (Zerfaß, 2005). An add-on to TMX called Segmentation Rules eXchange (SRX) facilitates the exchange of segmentation rules according to which translation units are created from longer texts during alignment. Information about these rules is necessary because leveraging of TMs is hampered when the tools create different translation units from the same text.

[32] LISA shut down on 28 February 2011 (Lingotek, 2011).

Other file format standards are TermBase eXchange (TBX) for terminology data, Open Lexicon Interchange Format (OLIF) for lexical data, and Global information management Metrics eXchange (GMX), a family of three standards, GMX-V, GMX-C and GMX-Q, for localisation-related metrics regarding volume, complexity and quality (Lewis *et al.*, 2009).

A special case is the Internationalization Tag Set (ITS) standard, defining XML tags that can be included in any XML file to provide metadata, context and translation instructions for content in other XML formats.

Many open standards lack the adoption in current tools, and the situation is of course worse for legacy software which will never adopt these standards. However, development for open standards can be comparatively uncomplicated. A good example is given by the Work in Context System by Bikmatov *et al.* (2013), which uses existing technology to display source text from XML or XLIFF files along with context information, metadata and translation instructions in a browser.

2.7 Software Localisation Practice

Only a limited number of comprehensive studies in software localisation not restricting themselves to specific aspects or contexts are extant. Some descriptive publications regarding internationalisation and localisation practice exist. For example, Hudson (1997) describes design, organisation and localisation of seven major software companies. Jin (1997) describes the implementation of a word processor based on the API framework outlined by Lehtola *et al.* (1997) and the internationalisation architecture described by Kokkotos and Spyropoulos (1997a, 1997b).

A more analytical study is given by Law (2003), describing internationalisation and localisation of a brokerage platform for some European locales including a survey of translators which reported demand of effort and time, and lack of suitable software tools.

Localisation and internationalisation practice has also been part of research that has a somewhat different scope. While examining consistency in translation memories, Moorkens (2012a, 2012b) conducted interviews with translators, managers and engineers on localisation processes as a means to triangulate his research results and identify inconsistency sources.

Immonen and Sajaniemi (2003a) conducted semi-structured interviews with professionals in management roles at six software companies and four LSPs from Finland to determine current practices and problems in software localisation with a focus on communication and cooperation processes. They found that the main problem seemed to be insufficient and limited communication between software developers and language vendors.

Based on a study of eight software development companies, O'Sullivan (2001b)[33] described a generic software localisation process and its relation to the software development process. He reviewed five development phases: functional specification, design and implementation, quality assurance, beta testing, and rollout and code review and found a large degree of procedural consistency with only minor differences across all eight companies. Further, O'Sullivan investigated causes of localisation errors and tried to develop an understanding how software can be localised without introducing full-fledged bugs into the software projects.

Ongoing process and software improvements have reduced relevance of some of the details described by O'Sullivan. For example, Locale support of operating systems through APIs has dramatically increased. Nonetheless, O'Sullivan (2001b) gathered information on localisation and internationalisation practice including tools and encountered issues. The description of the software localisation process, its interplay with different stages of the software development process, the dependencies among the various stakeholders of those stages, and the overall impact on localisation bugs presents a comprehensive view into the complexity of software localisation.

Abufardeh and Magel (2009, 2010) and Abufardeh (2008) examined the impact of cultural concerns on software development and engineering.

2.7.1 Interdisciplinary Issues in International Software Development

What makes software localisation an interdisciplinary effort? Or asked differently, what exactly are the interdisciplinary aspects of the development of international software? Localisation combines the efforts of a multitude of disciplines (Zouncourides-Lull, 2011;

[33] Part of this research has been published in O'Sullivan (2001a) and O'Sullivan *et al.* (2003).

Sturm, 2002; O'Sullivan, 2001b), but the user sees the resulting product as one (Collins, 2002).

The development of international software involves professionals from different disciplines, but this in itself is not uncommon. It is simple to characterise the interdisciplinary nature of the development of international software merely through the participation of professionals from different disciplines, here, translators and software engineers. But what makes development of international software complex is the need for multidisciplinary knowledge and interdisciplinary communication during localisation and internationalisation activities.

The need of cultural knowledge for the development of international software is rarely stated as openly as by Abufardeh and Magel (2008b) and Ryan *et al.* (2009). More often, it is implied, e.g. through the need of software engineers to identify and understand culture-related requirements (Abufardeh and Magel, 2010; Mahemoff and Johnston, 1998). Lack of cultural knowledge leads to what Vatrapu (2011) calls *ethnocentric assumptions*, i.e. the assumption that despite cultural differences, members of different cultures nonetheless come to the same conclusions judging presentation and functionality of software. Localisation and internationalisation also require technical knowledge, i.e. about concepts such as locale (Sikes, 2011) and usability (Russo and Boor, 1993), and about each other's discipline.

The need for multidisciplinary knowledge further implies the need to be able to transfer this knowledge across disciplines through communication (Bauer and Rodrigo, 2004). As Zhou (2011) points out, the need to communicate and coordinate between translators and engineers becomes more important the more complex localisation becomes, particularly if localisation is to accommodate a user's cultural characteristics and will produce different look and feel, functionality and usability, as discussed in section 2.4. Concluding her examination of use of localised products in target locales compared to the original locale, Sun (2004b, p.9) writes:

> *We need to have an expanded vision of localisation process [...] The scope of localisation should go beyond a single stage in the software design and engineering cycle (for example, translation and interface design) and enter the site of local use and consumption.*

This touches on another aspect – whether localisation is separate from software development, as described by some authors (e.g. Wahle, 2000), or part of it, as described by others (e.g. Collins, 2001; Hogan *et al.*, 2004). While Immonen and Sajaniemi (2003a) found that software developers consider software localisation separate of software development, against intuition they also found that software developers considered localisation a software developer's assignment and were likely not to outsource localisation unless they lacked time or language skill:

> *L10N seems to be considered purely as a [software developer's] assignment, and [LSPs] are used only if [software developers] cannot localise all the components themselves.* (Immonen and Sajaniemi, 2003a, p.161)

The authors concluded that this might be one of the reasons why LSPs are only contacted towards the end of the development lifecycle, i.e. when time become short, and further concluded that this might be the reason why LSPs are under time pressure.

2.7.2 Cultural Knowledge for Software Developers

The importance of cultural knowledge, or cultural awareness, has been acknowledged both implicitly and explicitly by various authors. Implicit acknowledgement comes in the form of calls to support developer access to cultural knowledge, often in the form of a document or database. Smith *et al.* (2007) call for the development of databases of local knowledge to help the developer. Carey (1998) suggests to have a so-called international functional requirement document and an international guide for programmers and writers. Mahemoff and Johnston (1998) propose a classification of cultural factors relevant for software, which they propose as base for a repository of cultural information, to be accessed by developers in order to identify and address culture-specific requirements. However, Collins (2001) warns that rules could lead to stereotypical views misrepresenting cultural richness and variation.

Alternatively, there are calls specifically of cultural education for developers. Hogan *et al.* (2004) calls for internationalisation and localisation aware developers, and Carey (1998) categorically states that all team members need to know about internationalisation issues.

The strongest argument, however, comes from authors who find that developers are required to have cultural knowledge as a direct consequence of the requirement to internationalise software (e.g. Liem *et al.*, 2011; Ryan *et al.*, 2009; Abufardeh and Magel, 2010). Particular emphasis is placed on the statement by Carey (1998, p.449):

> *Both internationalization and localization require that the programmers be aware of their own culture, language, social values and expectations. Localization requires more rigor than does internationalization. This is because localization teams tend to discover problems left by the internationalization team, but localized products move straight to the market upon completion so there is no downstream channel member to detect problems.*

2.7.3 Contrasting Engineers and Translators

It is conceivable that issues in software localisation can be attributed to the collaboration of professionals working in different disciplines, i.e. translation and engineering. In software development, the impact of distinctness of people has already been identified as a potential cause of issues. For example, Quintas (1993) observed differences between developers and users, who "tend to inhabit different physical spaces, have different career paths and reward systems, organize work differently, and employ different specialized vocabularies" (Quintas, 1993, p.5). The same criteria apply for developers and translators.

Existing literature and research has examined and characterised both software engineers and translators. Software engineers possess a trial-by-error mentality (Green, 1994) and have been attributed with a unique personality profile (Beecham *et al.*, 2008; Capretz, 2003) motivated by job aspects, e.g. technical success and challenging problems, rather than conventional motivating factors such as rewards and recognition (Beecham *et al.*, 2008). Cooper (2004, pp.93, 106) speculated that software engineers prefer control to simplicity, accept to pay with failure for understanding, overemphasize theory over practice, and offend easily.

On the other side, translators have been characterised as craftsmen with a vocation rather than a job (Sikes, 2011), a partner conducting cognitive work to aid in the creation of a product (Stoeller, 2011), or "nurturers, helpers, assistants, self-sacrificing mediators" (Pym, 2008, p.323) to their clients. As a consequence, translators are in a subservient

position. Bauer and Rodrigo (2004) attest translators an awareness of cultural differences and communication requirements, yet also technological literacy and resourcefulness. They are excellent communicators, but also risk-averse due to a lack of certainty in their work (Pym, 2008). Their role is generally understood to require domain-specific knowledge in the area of their source text documents (Hubscher-Davidson, 2009), they also need assistance by domain-specific natives (Sikes, 2011). Szuki (1988) found that translators are patient, with a primary interested in art and intercultural contact.

There have been efforts to understand particular characteristics of software engineers and translators through psychometric measurements such as the Myers-Briggs Type Indicator (MBTI), developed in the 1940s based on Carl Jung's theory of psychological types (Briggs Myers, 1962). The MBTI is based on the premise that individuals can be characterised through the dimensions extroversion (E) versus introversion (I), sensing (S) versus intuition (N), thinking (T) versus feeling (F), and judgement (J) versus perception (P). MBTI types are coded with letters according to their dimensional preferences, i.e. ISTJ would indicate an individual preferring introversion over extroversion, sensing over intuition, thinking over feeling, and judgement over perception. However, the MBTI types describe an individual's preferences and predispositions, not aptitudes.

Based on earlier research that suggested that introversion, thinking and judging are predominant characteristics among software engineers, Capretz (2003) conducted MBTI tests among software engineers and postgraduate software engineering university students (n = 100). The study found a clear relationship between psychological types and software engineers, with NT and ST types being overrepresented and ISTJ being the most common type among their participants, described as being technically oriented, preferring to work with facts and reason rather than people. It was concluded that

> *[T]he software field is dominated by introverts who typically have difficulty in communication with the user. […] [Software engineers] tend to be poor at verbalizing how the task affects the people involved. In fact, the greatest difference between software engineers and the general population is the percentage that takes action based on what they think rather than on what somebody else feels[34]. (Capretz, 2003, p.214)*

[34] Gladwell (2015) relates this characteristic as joke about engineers: While playing golf, a priest, a doctor and an engineer are frustrated by a group of firefighters ahead of them progressing very slowly. After

The MBTI was also used by Hubscher-Davidson (2009) to analyse translator's personality traits. The study reported a correlation between intuitive personality types and high translation quality, showing a skew towards introvert, feeling and judging types. Hubscher-Davidson (2009) did not aim to generate a personality profile of a professional population and did not contrast translator personality types with those of the general population. The most prominent difference between personality types of software engineers and translators is on the thinking-vs-feeling scale, where software engineers tend to be thinking types and translators tend to be feeling types. Tsvetkov and Tsvetkov (2011) have speculated on how to improve communication in localization project management on the basis of the MBTI and associates a thinking-vs-feeling divergence with a clash of factual versus emotional arguments during problem solving and conflict resolution.

It must be mentioned that Tsvetkov and Tsvetkov (2011) did not provide any empirical findings supporting their conclusions. In fact, although the MBTI has become one of the most widely used personality assessment tools and despite its use in research, it has been seriously criticised for a number of methodological shortcomings. Among others, it has a low re-test reliability, meaning test scores are not time-invariant and there is a high probability of obtaining different results from two tests of the same individual. Further, distributions along the dimensions are not bi-modal, meaning that distinctions between the two extremes of each dimension are statistically not warranted (Pittenger, 1993).

2.8 Summary

In this chapter, the need to adapt international software for locales, including scope and complexity, was demonstrated through results of empirical research. The activities localisation and internationalisation were introduced, and their interdisciplinary and multidisciplinary aspects were elaborated. Further, localisation issues were characterised and the interdisciplinary character of the development of international software was examined.

learning that all of the firefighters have lost their sight while saving the golf clubhouse from a fire, the priest states, “I will say a prayer for them tonight.” The doctor states, “Let me ask my ophthalmologist colleagues if anything can be done for them.” The engineer asks, “Why can’t they play at night?”

The activities of internationalisation and localisation can impact an international software product along the dimensions of cost, quality and time. Localisation can be incomplete or inappropriate, and both internationalisation and localisation activities can incur significant cost as well as delay product releases. These issues have been shown to be non-trivial and are conflated in the term localisation issues.

The particular interdisciplinary character of the development of international software has been evidenced in the interdisciplinary character of localisation issues: First, knowledge of disciplines, e.g. engineering and translation, is required to create international software. Second, the roles are intertwined, e.g. software engineers need to identify and understand information about locales, i.e. cultures, and translators need to identify and understand aspects of software development. And third, localisation issues create differing concerns for each involved discipline, just like the two disciplines differ from each other.

This chapter also gave an overview over research and engineering efforts undertaken with the aim of improving software localisation. Among others, the Unicode standard has provided widespread availability of the most common scripts. Equally widespread Unicode and locale support on the operating system level, the provision of internationalisation UI APIs, and the availability of internationalisation frameworks and standard architectures have greatly simplified the engineering effort for internationalised applications. Translation editors supporting both binary and source files, terminology databases, translation memories and other software tools have simplified the translation activity while allowing multiple translators to work on one project and helping to avoid redundancy in translation and to increase linguistic consistency. Finally, translation outsourcing has made localisation into multiple locales affordable even for small software companies.

These are only the most prominent developments and approaches, and some are ongoing efforts, e.g. encoding of the world's scripts in Unicode, improving MT, or the development of standards. Through such efforts, creating international software has certainly become easier. However, at the same time existing research is profoundly limited: The majority of existing research either revolves around a locally contained context of localisation and internationalisation, or proposes an approach to improve

localisation outcomes based on an architecture model. There is little research on internationalisation and localisation practice and general causes of localisation issues.

To be clear, the development of technical standards, tools and architecture models for localisation has certainly improved the state of development of international software immensely. Equally, research on what earlier was termed locally contained context of software localisation, e.g. tool use and further automation, is useful and constructive.

My argument is rather that notwithstanding these achievements, there is current practice and the underlying presupposition of a separation of software engineering and localisation which needs to be examined, but is really under-researched. I find this view supported in publications of a number of researchers in the area of localisation: Lenker *et al.* (2011) noted a focus in localisation literature on tools, technology and individual activities, rather than general workflow across the activities, manifesting itself in a lack of standard workflows for translation and localisation. Sasikumar (2004) suggested to conduct comparative studies of different types of localisation efforts, and the integration of localisation into the software development process is frequently discussed (e.g. O'Sullivan, 1989; Russo and Boor, 1993; Rafii and Perkins, 1995; Collins, 2001; Forssell, 2001; Hogan *et al.*, 2004; Abufardeh, 2008).

Chapter 3 Research Methodology and Method

The previous chapter reviewed existing related research and literature, leading to the formulation of research aims and objectives as discussed in sections 1.4 and 1.5. The research questions were as follows:

1. How is localisation conducted individually and collaboratively by developers and localisers, and how does this shape each discipline's activities?
2. How are issues caused during localisation and internationalisation?
3. In what regards are developers and localisers distinct?
4. What dependencies exist between localisation effort and properties of development projects?

The research methods must fit to the subject area and research questions (Robson, 2011). In this chapter, the choice of research methods is reviewed and documented. Section 3.1 discusses qualitative and quantitative research. Because both paradigms are applied, mixed methods and their applicability are discussed in section 3.2. Sections 3.3 and 3.4 discuss and justify the chosen qualitative and quantitative data collection and analysis methods. Section 3.5 reviews sample and population used in this research, and section 3.6 notes applicable ethical considerations.

3.1 Qualitative and Quantitative Research

Qualitative research methods aim to integrate complexity. They are often used in the study of social relations and human behaviour (Flick, 2002; Seaman, 1999). Use of qualitative research methods acknowledges the complexity of situations that cannot be stripped down to trivial and unambiguous cause-effect relationships. Subjectivity of participants and perspectives, construction of reality and reconstruction of data, and a reflection on the research process as well as the researcher himself feature heavily in qualitative research (Flick, 2002). Qualitative analysis relies on finding context- and subject-specific descriptive and exploratory schemes in data through interpretative discovery of relationships and concepts, referred to by Strauss and Corbin (1998) as conceptualising, reducing, relating and elaborating. The data is reduced and visualised, and concluded in the form of regularities, patterns, explanations and propositions (Miles and Huberman, 1994).

Reduction forms a central part in qualitative data analysis and is often done through coding. Coding is a largely creative process during which the researcher tries to express the conceptual meaning of a section of data, for example a single event or a series of events during observation or a section of text in an interview transcription, as objectively as possible through a designator consisting of a few key words, a so-called *code*. Codes can be pre-formed, i.e. decided on before coding if the objectives of the study are known beforehand, or post-formed, i.e. derived from data in the case of open and unfocused studies (Seaman, 1999).

Quantitative research methods aim to examine a subject as objectively as possible by reducing complexity and eliminating subjectivity, e.g. observer and selection biases. Subsequently, while the research results should be reproducible, their practical value is limited because objectivity requirements impact practical applicability of results (Flick, 2002). For example, in a laboratory experiment any potentially influencing factors have to be carefully controlled in order to ensure that any differences between treatment and control group can be ascribed to the intervention, i.e. a change in an independent variable. But many phenomena with interest for practice occur only in complex situations which cannot be replicated under laboratory conditions or comprehensively recorded through quantitative means.

Quantitative research often aims to test associative and causal relationships through calculation of a so-called *significance level*, also referred to as *p-value* (Field, 2005). The p-value is often interpreted as the probability that a research result has been arrived at by chance, although this is only correct under specific circumstances. It is better interpreted as an arbitrarily chosen value to differentiate research results based on their mathematical strength (Nuzzo, 2014; de Groot, 2014).

Accordingly, qualitative and quantitative methods follow different agendas. Qualitative methods are mostly used in exploratory research, to examine social relationships and phenomena within a specific context about which not much is known, or when an interpretivist or constructivist paradigm applies and what is considered reality is constructed by individuals or dependent on their interpretations. Quantitative methods are mostly used in explanatory research, to confirm already existing theories about a

subject, or when a positivist paradigm applies and reality is considered to be objective and measurable (Flick, 2002; Leedy and Ormrod, 2013).

3.1.1 Mixed Methods

The combination of qualitative and quantitative research methods in a single study is called *mixed methods*. It is considered to be more effective than relying on only one method (Seaman, 1999; Runeson and Höst, 2009).

A number of empirical software engineering studies employ mixed methods. For example, Linberg (1999) explored how software developers define success and failure of a project, how failure affects job satisfaction, and how failure was related to individual developers' temperament. Linberg combined qualitative analysis of interviews and project documentation, and statistical survey analysis. Espinosa *et al.* (2002) used a sequential approach, i.e. quantitative examination of phenomena via survey from previous qualitative interview and archival research, to show how shared mental models, work familiarity and geographic dispersion benefit coordination in software teams and shorten development time.

Strictly speaking, it is not application of both qualitative and quantitative methods or gathering of both qualitative and quantitative data that makes mixed methods research. What really is required is that the data works in conjunction, e.g. as sequential approach deriving a framework qualitatively and then testing it quantitatively, or as methods triangulation by using both qualitative and quantitative methods to answer similar research questions and increase result validity (Creswell and Clark, 2007).

In this research, qualitative and quantitative methods are applied in isolation to answer separate research questions. Hence, although both methods' results might be combined, e.g. by comparing descriptive qualitative results with descriptive statistics, the mixed methods moniker is not appropriate.

3.2 Using Grounded Theory to Explore Software Localisation

The literature review suggested that software localisation is conducted in a social context of software development and is affected by human factors. For such phenomena which are difficult to study in isolation, qualitative research is the appropriate approach (Runeson and Höst, 2009). Hence, RQ1 and RQ2 are best addressed by qualitative

research methods. The part of research that can be examined through testable hypotheses is separated out. The question of generalising qualitative research results is mitigated by the common view that software development projects are so diverse that generalisation is difficult anyway; a notion confirmed by the views of software developers themselves (e.g. Umarji and Seaman, 2005).

To examine how localisation is conducted, how this shapes the activities of localisation and internationalisation, and how it causes localisation issues, Straussian Grounded Theory is used. The next subsection will illustrate common research methods in empirical studies of software development, justify the choice of GT, and discuss its application.

3.2.1 Selecting Qualitative Methods

This subsection reviews the choice of data collection and analysis method. The collection method must yield data that can answer the research questions and must fit to the remaining research method, i.e. analysis. The analysis method must fit format and subject of collected data, the research questions, and the intended outcome.

Some methods were dismissed outright. For example, action research, where the researcher joins a team in trying to solve a problem while taking notes for later analysis (Christiansen, 2010) was deemed to be unsuitable because it is presumably difficult to find software companies allowing this kind of access, and further iterations in iterative analysis methods take too long.

Empirical studies of software engineering often use archival data, observations and interviews for data collection, and template analysis, framework analysis or GT. For each, advantages, disadvantages and fit to this research are discussed.

3.2.1.1 Archival Data

In archival data studies, archival evidence is studied, i.e. existing artefacts created by the organisation one wishes to examine, possibly created during or for the activity one wishes to study. These artefacts usually come in the form of some kind of documentation.

Empirical studies of software development seem like a good fit for archival data studies because software development by necessity involves the creation of artefacts with additional documentation that often is comprehensive and well maintained. For example,

it is best practice to store source code in repositories in order to record who conducted what changes at what time. Similarly, bug tracking databases keep track of program errors and how they were handled. Further information might come from test protocols, email exchanges between developers and with customers, and requirements documents. Such documentation is regularly used for studies. For example, one of the data sources for the research of Grinter (1995, 1996a) on the use of configuration management tools were documents and logs of discussions among developers conducted via electronic media. Likewise, Espinosa *et al.* (2002) supplemented the study on the effect of shared mental models with archival sources. In the area of localisation, Moorkens (2011, 2012a, 2012b) used TM databases for his research on consistency.

Archival data is a tertiary data source, meaning that it has not been collected for research purposes (Runeson and Höst, 2009). It allows a view on what happened, but is usually restricted in the information it can deliver because it only contains information related to the purpose it has been collected for. The effort to extract research-usable information from archival data can be considerable.

In the case of this research, the main archival data sources I expected to find about usual localisation work were source code repositories including change logs, bug tracking databases, TMs, software design documentation, and maybe communication protocols between engineers and translators. There are two reasons why the first four sources are unpromising, though. First, they likely contain information on what has been done, but not how and why. Second, each likely only contains input from either engineers, or translators, but not both. As for communication protocols, their very existence is somewhat speculative as a number of literature items in chapter 2 suggest that communication between engineers and localisers was lacking. For these reasons, archival data was dismissed as data source for this research.

3.2.1.2 Observation

In observation methods, the researcher directly observes the object of study. A variant of observation is so-called *participant observation*, where in addition to the observation sessions data is collected during interactions between the observer and participants (Seaman, 1999).

The advantage of the observation method lies in the directness of data acquisition. Interviews and surveys require participants to reflect on their activities after the fact, which brings the danger of them rationalising their actions upon reflection. For example, an amethodical progression of activities might be reported as a methodical process (Truex *et al.*, 2000). In fact, e.g. Winter and Rönkkö (2010) noted that many activities in software development are not planned and conducted as rationally and sequentially as reported.

Because observation comes with the advantage of giving the researcher direct and unfiltered access to the phenomena under study, it is a strong method for studying people's behaviour and interactions, especially when there is a reason that an unbiased account of actual events could not be obtained from participants (Kvale, 2007; Runeson and Höst, 2009). Subsequently, observation and participant observation seem to be very popular methods in empirical studies of software development. For example, Grinter (1995) examined the use of a configuration management tool in organisations, specifically how the tool integrated into collaborative interactions between developers through analysis of observations, interviews and archival data. Plonka *et al.* (2011) examined the switch between active and passive developer during pair programming through screen capture and video recording of developers. Ferreira (2011) examined the combination of agile software development methods and UX design through observation and interviews. Participant observation in particular seems to be popular in GT studies because they allow method triangulation for theory verification, and has been used to that effect by Hoda (2011), Martin (2009) and Abdelnour-Nocera (2007).

Observational studies come with a number of disadvantages and limitations, though. Observation works best if at least two researchers can compare their observation to ensure completeness and objectivity (Seaman, 1999). It requires considerable amounts of time (Hoda *et al.*, 2011) and produces large amounts of data (Runeson and Höst, 2009), reducing the number of cases that can be studied and analysed. Further, not everybody is comfortable being observed, so it might be more difficult to find participants for observation studies. This limitation should be considered particularly for GT studies, as the method triangulation afforded by participant observation might come at the price of decreased data triangulation.

Another difficulty is that opportunities for meaningful observation in software development are limited, e.g. to meetings and exchanges (Seaman, 1999). A major part of software development is individual cognitive work, often conducted in relative silence in front of a computer. Remedial devices such as think-aloud protocols are a considerably disruptive cognitive effort in this context.

The act of observation itself must also be considered as an influence on the participants and activities (Seaman, 1999). The most famous example of this might be the *Hawthorne effect*, also known as the *observer effect*: In an industrial study examining the relationship between lighting levels and worker productivity, it was found that productivity increased regardless of the direction of change in lighting, but decreased back to normal after the study had ended. The productivity increase was attributed to motivational effects stemming from attention of researchers (Landsberger, 1958). The Hawthorne effect and its interpretation has since been examined critically (e.g. Adair, 1984), but there is a general agreement that observation can affect participant behaviour. Hirschheim and Klein (1989, p.1204) phrases it more aggressively:

> *People have free will and observation is not neutral. […] [P]eople as objects of study always 'observe back'. They can perceive the observer's plan of study and counteract it.*

Nonetheless, it was initially considered to conduct observation or even participant observation due to these methods' strength to collect unfiltered data on social interactions. Contact with a few software companies had already been established, but negotiations over the level of access highlighted a particular difficulty with observation studies on software localisation.

The work of engineers and translators in front of a computer does not lend itself to observation because most of the work is cognitive and will not leave visual clues, as discussed above. This leaves interactions of engineers and translators to be observed. Those are likely to be unplanned and unscheduled[35].

[35] The point can be illustrated by examining the use of the observation method in Plonka *et al.* (2011): Pair programming is a continuous activity. The study is mostly interested in the switch of mouse and keyboard control, events that are likely to happen several times in a session. So, each observed session adds useful data. In software localisation, this is different. My experience, not disputed by the literature review, suggests that it is difficult to know beforehand if any meeting will discuss localisation or a translator's or engineer's work day will involve communication with the other discipline, if it takes place at all.

Hence, pure observation must be supplemented by additional data. Since archival sources would require more access negotiation while likely yielding only limited improvement as discussed in the previous subsection, the next obvious alternative would have been a combination of observation and interviews, or participant observation.

This led to a critical view on participant observation based on studies and methodology literature. The idea behind participant observation is to use interviews to gain additional insight into observed events. However, it appeared to me that in a number of participant observation studies, the observation cues the interview. Or to put it bluntly, the observation serves as a social situation for the researcher to arrange a chat later. I say *chat* because often, the following interviews are informal and short. Additionally, most of them seem to be only weakly motivated by any observation, but nonetheless lack an overarching interview strategy. In other words, participant observation requires extreme discipline when conducting the interviews because by the nature of participant observation, preparation for and conduct of the interviews is limited. Similar criticism of participant observation is discussed by Hammersley (1992) and Haralambos *et al.* (2013).

To that effect, participant observation can easily lead to unwarranted notions of informality in both researcher and participants. Since observation alone is unlikely to yield relevant data and participant observation seriously limits interviews, it is preferable to conduct interviews separately and drop observations and the disadvantages that come with it.

3.2.1.3 Interviews

In interviews, data is gathered through direct conversation between researcher and participant. Interviews can provide an almost arbitrary depth, provided the interview is not structured, as it is easy to engage with and react to responses from participants. Interviews also present an opportunity to obtain data which is not accessible through observation, e.g. attitudes and dispositions of participants. In face-to-face interviews, non-verbal cues can help to gain an understanding that not all other data gathering methods offer. Further, multiple interviews can help to obtain a wide range of phenomena while maintaining replicability and generalisability (Salo and Abrahamsson, 2004).

Interviewing means having to consider unreliability of retrospective accounts on past events. Generally speaking, there is a difference between what people do, and how they describe it later (Paetsch *et al.*, 2003; Truex *et al.*, 2000). In particular, Perry *et al.* (1996) found that the accuracy of reporting the short and unplanned events and details is overestimated. Interviewing can be a very personal affair for both interviewer and participant, and due to the time it requires of the latter, recruiting can be difficult. For the interviewer, it can be time-consuming compared to some research methods, e.g. surveys, but time-saving compared to others, e.g. observation.

For this research, interviewing was chosen because it enabled the collection of both accounts of practice and of insights into participants' thoughts and opinions. Interviews have the potential to reveal what archival accounts cannot: what has been done, how it has been done, and why it was done the way it was done. Compared to observation, the data limited in interviews is moderate and analysis affordable. Regarding the validity of accounts, I chose the discrepancy between participants' accounts and what really happened over the discrepancy of observed behaviour modified by researcher presence.

3.2.1.4 Template Analysis

In template analysis, prior to coding, a coding template is created with pre-formed codes that the researcher expects to be important in the data. The pre-formed codes are then identified during initial coding of a part of cases. Should new themes become apparent, or if pre-formed codes turn out to be unimportant, the coding template can be modified accordingly before continuing with further cases. This way, the template is developed during coding until, after coding is completed, the final template can be used to write up findings and interpretations.

One of the advantages of template analysis is that pre-formed codes can speed up coding, and that the researcher has some level of control over the codes, for example if specific themes in the topic are already known, or are the subject of the examination, e.g. in the context of an evaluation. On the other hand, the more pre-formed codes have been defined in the initial template, the more likely is missing important themes for which no pre-formed code exists.

Template analysis is often used for interview data and is considered a suitable analysis method for software engineering case studies (Runeson and Höst, 2009). For example, Zhang (2012) used template analysis for the analysis of interview records in order to review tools and practice in prototype design and potential use of virtual worlds for the early building construction process.

For this research, template analysis was discounted due to its focus on pre-formed codes. Because the choice had already been made to examine differences between developers and localisers, role of cultural competence, and influence of project properties on localisation through a quantitative survey, it was considered preferable not to guide the qualitative research through preconceptions expressed in pre-formed codes.

3.2.1.5 Framework Analysis

In framework analysis, familiarisation with the topic is followed by the choice of a thematic framework, which is then systematically applied to the data during coding. The framework is further used during a subsequent stage called *charting* to abstract, order and synthesise the data so that concepts leading to an interpretation can be created.

Despite being a deductive method, some authors have suggested that framework analysis is quite similar to the eventually chosen GT, but more suited to answering specific questions and examining already identified issues (Srivastava and Thomson, 2009). It is a common method in computing, for example Abdelnour-Nocera and Sharp (2008) used framework analysis of technological frames of Bijker (1997) to confirm the importance of pan-organisational consultation and specific work-step explication during the adoption of agile software development processes in large organisations, and further to examine the cultural dependence of the usefulness in the context of enterprise resource planning systems (Abdelnour-Nocera *et al.*, 2007; Abdelnour-Nocera, 2007).

Framework analysis was initially considered as analysis method, but eventually the lack of a convincingly suitable framework as well as the research restrictions of any framework, similar to those of the template method, suggested that a method requiring no underlying existing theory is preferable for this research.

3.2.2 Grounded Theory

GT aims to identify and categorise elements and explore their connections (Miles and Huberman, 1994). It is based on the premise that interpretation of data is more important than the way in which it is gathered. GT is used for exploratory and descriptive research and aims to generate theory, i.e. "systematically interrelated categories with explanatory power" (Strauss and Corbin, 1998, p.20) and descriptions, the latter of which can lead to conceptual order, i.e. organisation of data based on properties or dimensions. The research process in GT is iterative, i.e. findings are used at all stages in the process and eventually lead to a refinement and narrowing of ongoing research.

GT is particularly useful in research areas in which previous research has been limited (Hoda *et al.*, 2011), as is the case with collaboration in localisation. As a qualitative method, GT has the advantage of examining phenomena in context and does not require a reduction in complexity to work. Due to its iterative nature and the feeding back of findings into the research process, it is a good method for examining vaguely known areas (Hoda *et al.*, 2010).

The drawbacks of GT are a susceptibility to researcher bias, i.e. a researcher's preconceptions and assumptions can easily influence the outcome. Further, GT is a relatively cumbersome process as it affects almost all areas of research, including participant sampling and writing up (Hoda *et al.*, 2011). The literature review should be light to aid in the avoidance of biases. It has also been noted that GT aims for two contradicting goals: the examination of situations in context, and the generation of abstract theories with the aim of eventually applying them outside of their original context (Haralambos *et al.*, 2004; Hammersley, 1992).

GT is most often applied in sociology and nursing (Adolph *et al.*, 2011). In recent years, it has become more popular in computing-related studies examining social aspects of computing and software development. Coleman and O'Connor (2008) examined the formation of software development processes as a function of best practice models and cost in 21 Irish software companies. Dagenais *et al.* (2010) examined the experience of 18 newcomers when joining already existing projects, and concluded that successful integration depends on the newcomer's experimentation with, and acceptance of, existing project structures and cultures, and feedback on integration progress by the

existing project team. Crabtree *et al.* (2009) conducted GT in a treatment setup to explore the dependence of software process description on perspective and context of four participants. Hall *et al.* (2009) examined the non-adoption of localised software in Nepal, attributing the lack of success of Nepali-localised software to issues with the actual software interface and a socio-economic environment which promotes the use of English software.

GT has also been used in a number of PhD theses in computing. For example, Grinter (1996b) applied GT for examining how software development organisations manage dependencies in the production of software systems and their technical and social aspects. The researcher examined the concept and nature of dependencies, why they occur, and how developers and organisations cope with them. Martin (2009) used GT to examine customer role, experience, and its role in the requirements elicitation process in extreme programming (XP) projects. Hoda (2011)[36] created a descriptive grounded theory of self-organising agile teams, consisting of roles, practices and factors.

GT was originally developed by Glaser and Strauss when studying the sociology of dying (Glaser and Strauss, 2009). Although initially developed in collaboration, interpretations of GT soon forked into two major camps[37]: That of Glaser, and that of Strauss and Corbin. Glaser (1978) continued to develop the concepts of theoretical coding, sampling and memos, while Strauss and Corbin (1998) adapted GT to make it easier to use. Accordingly, when applying GT, a decision has to be made what particular approach to follow. This is important because while the two approaches are generally similar (Adolph *et al.*, 2011), care must be taken not to mix aspects that are not compatible, particularly when using auxiliary literature that either does not specify which approach it used, or that used the other approach. The major differences between Glaser and Strauss are philosophical, particularly regarding theory and theory generation and the part of induction, deduction and verification. For example, according to Glaser, in the initial coding phase the only coding technique to be applied is what he calls *substantive coding*. Strauss and Corbin on the other hand describe two techniques, *open coding* and *axial coding*, during the initial

[36] Partly published in Hoda *et al.* (2012).
[37] Further variations to GT are listed for example in Heath and Cowley (2004).

coding phase. Both Glaser and Strauss recommend so-called *in-vivo coding*, where the name of the code is derived from a term occurring in the data.

Heath and Cowley (2004) recommend to choose GT flavour based on the best fit to a researcher's cognitive style. I follow the GT approach of Strauss and Corbin for two reasons: Glaser's writing is arguably more difficult to understand. This might be a matter of taste, but some of the instructions border on the mystical. Intentions and meaning are often not as clear as in Strauss' and Corbin's instructions. Further, Strauss' and Corbin's approach seems more technical while at the same time being more tolerant towards the realities of research. Adolph *et al.* (2011) report that in software engineering research Strauss and Corbin seems to be the favoured approach.

3.2.2.1 The Research Process in Grounded Theory

As a research method, GT deviates in several ways from classic quantitative research approaches (Flick, 2002). GT is not a linear research process where each phase, i.e. theory formation, data collection or data analysis, is executed in isolation and finished before moving on to the next. Instead, GT is an iterative process in which the phases of data collection, data analysis and theory formation are repeated until the theory is not progressed by additional data any more. While quantitative research starts with a theory, then collects and analyses data, GT starts with data collection and analysis, which then leads to the derivation of theory. It is conducted as follows:

A case, for example a participant for an interview, is selected based on availability. Ideally, cases are selected to broadly cover the research subject. Data is gathered, e.g. by interviewing a participant, and then coded using open coding[38] and axial coding. During open coding, all the data is scrutinised for concepts through line-by-line coding. In axial coding, the concepts gathered in open coding are examined for dimensions which relate to their occurrence, e.g. causal and intervening conditions, context and consequences. All the concepts appearing in the data are coded. The codes are post-formed, i.e. they are derived from the data (Seaman, 1999). The aim is to find a structure relating concepts.

An important part of coding is the so-called *theoretical sensitivity*, i.e. a researcher's ability to understand subtleties of the data. Theoretical sensitivity can be attained from

[38] An interview sample including open coding can be found in Appendix C.

experience in the field, e.g. through repeated exposure to, reading about or working in the field. Theoretical sensitivity refers to conceptual knowledge about a research topic in order to help understand the data. It must be distinguished from concrete assumptions that might guide coding and influence the emerging theory and must be avoided in the GT process (Glaser, 1978).

Once coding of a case finishes, the next case is selected, data is gathered and analysed, and so on. Any theory derived from a new case is retroactively applied to all previous cases. This is referred to as *constant comparison*, meaning that information from incoming data is constantly compared to previously analysed data. New codes, derived from the latest case during open coding and thus post-formed, are applied to previous cases, and thus become preformed.

The researcher notes any theoretical insights gained during data analysis in so-called *memos*.[39] This process of selecting and analysing a case, constantly comparing new and previous cases and writing memos is repeated until a single *core category* emerges.

The core category refers to the central aspect of what is being researched. Once the core has emerged, the research process changes fundamentally: Open and axial coding are replaced by selective coding. Selective coding means that only those concepts are coded that relate to the core. The codes are still pre-formed, but must be guided by the core category. Constant comparison and writing of memos continues. Sampling should ideally switch from exploratory all-you-can-get sampling to theoretical sampling. Theoretical sampling means that selection of cases is guided by the analysis of previous interviews. Ideally, new cases are selected based on what the researcher assumes will expand the theory and aid representativeness (Seaman, 1999).

This continues until saturation is reached. Saturation is the moment at which new interviews merely fit into the existing theory without generating new knowledge. This is another deviation from classic tenets in research, which generally assumes that more data equates better scientific results. This is not true in GT. On the contrary, it has been pointed out that too much data might impede GT for practical reasons, e.g. because it makes a thorough analysis more difficult and eventually impossible (Kvale, 2007).

[39] An example of a memo is shown in Appendix D.

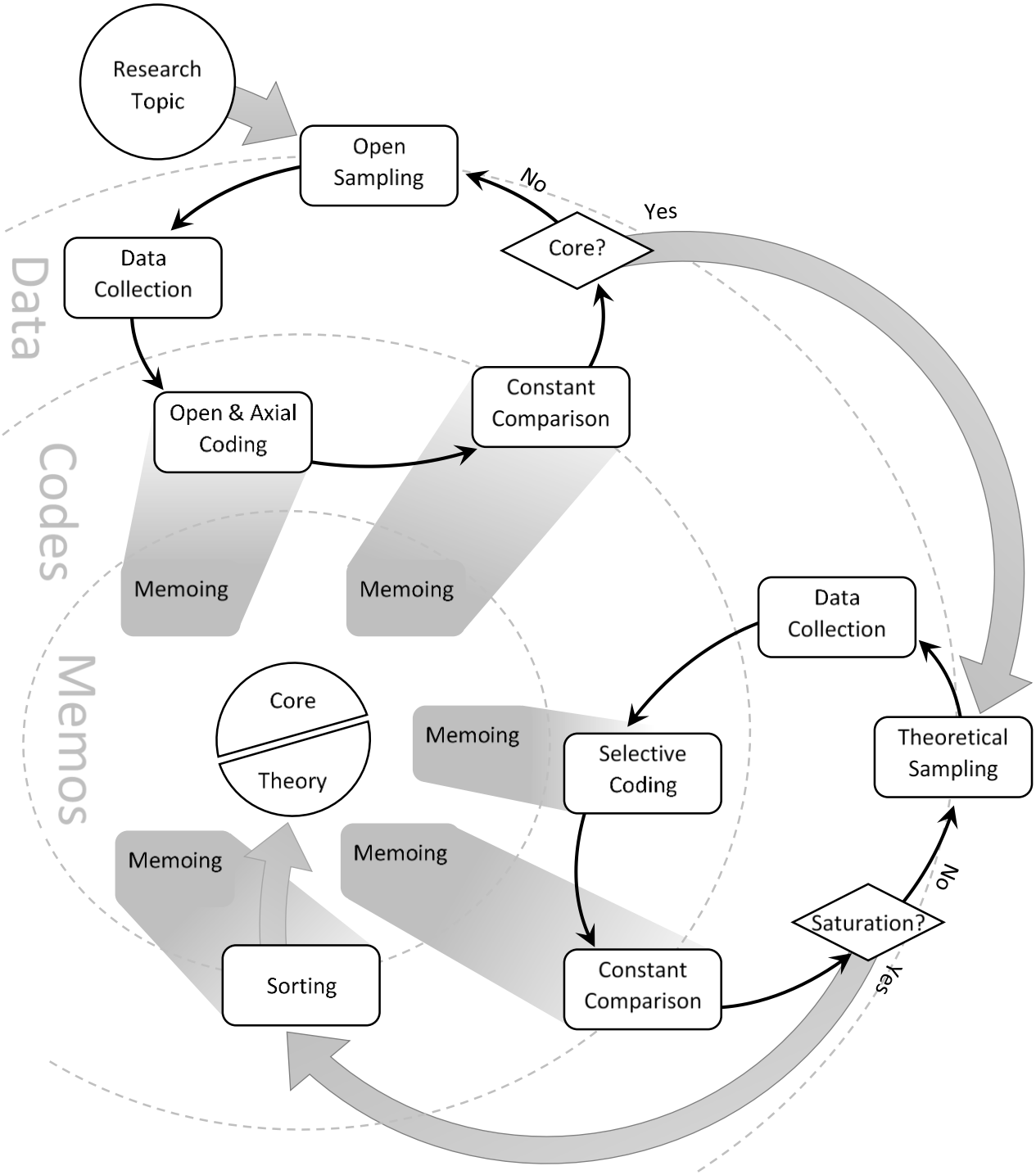

Figure 3-1 The research process in Grounded Theory

Once saturation is reached, codes, categories and memos derived from the data need to be connected and integrated into a theory. In GT, this activity is called sorting. During sorting, new ideas can still emerge which in turn have to be written into memos and integrated in the sorting process. Once sorting is finished, a rich and grounded theory should have emerged.

An interview excerpt is shown in Appendix C, a memo example is shown in Appendix D. The GT research process is shown in Figure 3-1. Strauss and Corbin (1998) concede that deviations from this process might be necessary. For example, research might have to progress to the final stages, i.e. theoretical sampling and eventually write-up, through lack of funding or time rather than the clear emergence of a core category. The authors also concede that theoretical sampling might not always be possible due to lack of access, in which case a take-what-you-can-get approach to sampling might have to continue.

3.2.2.2 Grounding Research – Validity and Reliability in Grounded Theory

Science needs to be linked to empirical verification and must be falsifiable in order to be able to claim finding new knowledge (Ellis and Silk, 2014). In classic research, this includes qualifying items such as validity and reliability of research results, but this is difficult for qualitative research and a topic of ongoing discussion. Flick (2002) suggests to supersede validity and reliability in qualitative research with alternative criteria: trustworthiness, dependability, credibility, transferability and confirmability,

Reliability indicates the absence of selection bias and the potential to reproduce results. This can, according to Flick (2002), be partially replaced by demonstrating adherence to standards, by training and practicing the methods to be applied, and by reflection and exchange about methods and interpretations. Further reliability can be achieved by comparing interpretations from one part of the text against other parts, or other texts. Finally, Flick (2002) recommends the documentation of procedures and data sources by specific separation of raw data and researcher interpretations.

Validity indicates the absence of interpretation bias. In GT, ensuring validity is called *grounding research*. Flick recommends to scrutinise interviews for strategic interests by interviewer and participant and for systematic influences shaping interviews and interpretations. Another technique is to ask participants for a review of the interview interpretations. However, this is problematic insofar as a dismissal of the interpretations by the participant does not necessarily mean that the interpretations were illegitimate. Some researchers apply quantitative interrater reliability tests to validate their concepts and categories (e.g. Gizaw, 2014). This was not done for this research as it requires recruitment of additional coders.

3.2.2.3 Previous Knowledge and Pre-Formed Theories in Grounded Theory

Strauss and Corbin (1998) explain that in GT, a researcher must be free to enter a subject area without prejudices and preformed concepts. So, the researcher must not know too much about the subject area he is conducting GT in. They do not define a particular amount of knowledge or a threshold excluding a researcher from applying GT to specific research, but point out that researchers should not have a theory. Instead, the researcher should be open to be guided by the data. The authors also recommend not to conduct too much literature review prior to data gathering and analysis.

A concern when choosing GT as research method was whether due to my previous occupation I already knew too much about software localisation to apply it effectively. I will address this concern below.

It is unrealistic to assume that a researcher has no knowledge about the area under research. Although Flick (2002) assume that a researcher looking into social relations is likely to be unfamiliar with the particular situations, that author also acknowledges that research questions are often related to a researcher's biography, as in my case. It is further recommended to define key concepts of research right at the start. Strauss and Corbin (1998) acknowledge that researchers have previous knowledge about their subject areas (see also Miles and Huberman, 1994). In fact, one would expect that a researcher has conducted a minimum of a subject area knowledge gathering before even being able to conclude that GT is a good choice for a particular research problem. Instead, I put forward that Strauss and Corbin (1998) mean to emphasize the exploratory nature of GT as opposed to explanatory research.

Research can be classified into four different types: Exploratory research, descriptive research, explanatory research, and improving research (Runeson and Höst, 2009). Exploratory research aims to generate new ideas and general understandings of new phenomena which have not yet been thoroughly understood. Descriptive research aims to describe and classify the state of things, e.g. characteristics or properties of phenomena. Explanatory research aims to find causal relationships, that is, mechanics explaining the occurrence of phenomena. Improving research aims to evaluate the often practical occurrence of phenomena in everyday situations.

As discussed earlier, Strauss and Corbin (1998) state that GT aims for exploratory and descriptive research. It has the objective of generating theory, e.g. in the form of hypotheses. It provides a data-based method for theory generation and liberates the researcher from generating for example classification schemes or hypotheses through a creative process *ex nihilo*. However, as GT is exploratory research, generated hypotheses are not verified. This is in contrast to classic explanatory research where an existing theory or hypothesis is confirmed or rejected based on comparing theoretical predictions to results of analyses of actual observations (Perry *et al.*, 2000). In de Groot's empirical cycle of observation, induction, deduction, testing, evaluation, *da capo*, theory generation corresponds to the first three phases (de Groot, 1969).

In other words, GT can generate theories, but not confirm them. This is because despite being data-based or methodological, it is not intended to be inter-researcher-reliable and does not follow classical notions of validity, reliability, and generalisability in research, which are anyway not considered to be appropriate for social research. Confirming or rejecting theories, generated through GT or otherwise, requires explanatory research.

Thus, the concern should not be how much knowledge a researcher already has in a subject field, but whether a researcher already has a theory about a problem. If there is a proposition what mechanics might be at play, what influences what, what moderates what, or similar, then there is no point in applying GT because it would either lead to the creation of an alternative, different theory, or the researcher again arrives at the original theory. Arriving at an alternative theory would merely create a new theory qualitatively different from the original insofar it was methodologically derived from data. Arriving at the original theory however would not confirm it, as GT is not a theory confirmation method. The original theory is still not validated and thus in no way, shape or form better, more substantial or more confirmed than it was before.

In other words: If a researcher already has a theory, there is no point in applying GT because it is redundant: it can only deliver what the researcher already has, an unconfirmed theory. Applying GT nonetheless might lead to a misguided notion that the original theory is confirmed or rejected through the result of GT, when it can actually not do any such thing.

The existence of a theory also stands in the way of theory generation. Such a theory or framework might lead a researcher to make assumptions about what is happening and what data would be most meaningful to collect (Miles and Huberman, 1994). In fact, there is research showing that cognitive processes, i.e. perception, attention, data interpretation, data production and memory, as well as scientific communication are theory-laden, i.e. influenced by theory. A review of existing studies was conducted by Brewer and Lambert (2001), showing among others that comprehension and memorisation is greatly improved when a theory or framework for information is available. In other words, a researcher is more likely to understand and remember data when it fits to an already existing body of knowledge. Incongruous data is ignored or forgotten, and thus data is superseded by theory. Uncannily, the effect increases the more ambiguous, complex, or degraded the data is. In other words, a pre-formed theory will inevitably influence a researcher's data interpretation, particularly when objective criteria are sparse, as in qualitative analysis.

In the light of the above, I argue that I am applying GT correctly despite any knowledge I already have. Previous knowledge itself does not matter. What matters is that I have no theory of what is going on in software localisation.

Now, one could say that I happen to have a theory: I am after all testing whether developers have a higher self-efficacy, but lower attitude and lower cultural competence in software localisation than localisers. This is true: yes, I have a theory, and I am testing it as best as I can. But this is independent of the interviews and my application of GT.

3.2.3 Application of Grounded Theory

I was conducting an exploratory, interpretive, flexible design, blocked subject-project case study using GT. I tried to understand phenomena occurring during the development of international software through interviewees' interpretations, adding to existing knowledge by building theory. The case is a process of developing international software, unit of analysis is the individual participant, with potentially multiple units of analysis per case. The method of data collection is direct. The selection strategy is both typical due to the request for participation, and revelatory due to theoretical sampling. Details of how I conducted GT are given in subsection 3.2.2.1.

3.2.3.1 Selecting Interviewees

For open coding, sampling of cases was determined by some *a priori* decisions: interviewees were selected when they had participated in the development of global software in a relevant role, i.e. translators, software engineers and project managers. While other roles, e.g. terminologist, reviser, DTP specialist, software tester and QA manager are mentioned in the literature (e.g. Hartley, 2009; Moorkens, 2012a), these were considered less relevant because they assumedly have less influence on internationalisation and localisation.

For the selection of initial interviewees, a convenience sample was deemed sufficient, following the recommendation "take what you can get" (Strauss and Corbin, 1998, p.208), explicitly allowing convenience samples for open coding. Because GT does not aim for initial generalisability of its results, it is of no concern if participant selection is not representative of the general population: open coding aims to find codes in context and generalisation of the codes is not intended. Once the core category has been found, further samples are ideally selected by theoretical sampling, ensuring appropriateness of the sample.

3.2.3.2 Conducting Interviews

The aim of interviewing was to understand the experience of professionals in the field, including meaning, feelings, value judgements, and of course accounts of fact.

Three types of interviews can be distinguished (Seaman, 1999). Structured interviews have pre-formulated and specific questions to be answered by the interviewee. In unstructured interviews, the interviewee is the source of both answers and questions, which are open and open-ended. In extreme cases, unstructured interviews resemble more a conversation than an exchange of questions and answers. For semi-structured interviews, a mix of specific and open-ended questions is used, the latter accounting for both foreseen and unexpected information emerging during the interview.

Strauss and Corbin (1998) recommend open questions during the use of GT. Their argument is that the more structure there is in interviews, the more likely participants are to answer the questions and nothing else. Less structure and open questions on the other hand increase the chance of participants soliciting experience and share what they think

is important beyond what is in the question. Further, commonality in interviews, i.e. structure across interviews, increases comparability and is considered more economic (Flick, 2002). Eventually, semi-structured interviews were chosen for two reasons: unstructured elements allowed to explore new areas and findings as interviews progressed. I believe this is an essential part of interviewing when conducting GT. Especially during theoretical sampling, interviews are strongly determined by previous findings. However, a certain amount of structure accounted for a minimum of relation of accounts from two comparatively different disciplines, software engineering and translation.

While there are few rules and standards for qualitative interviews (Kvale, 2007), Flick (2002) discusses five methods of semi-structured interviews, to be selected according to appropriateness for research topic and fit to research process: focused interviews, semi-standardised interviews, expert interviews, ethnographic interviews, and problem-centred interviews.

Focused interviews aim to analyse participants' perceptions of a common, specific stimulus, e.g. a known event such as a movie, and potentially to compare it with an objective analysis of the stimulus. This method was discarded since this would have severely restricted participant selection, e.g. having worked on the same project and experienced the same bug. No stimulus universal to the population of participants is known. Further, even if enough participants named one it was not actually desired to limit the research to a specific event or similar.

Semi-standardised interviews aim to understand participants' subjective theories about a topic through a series of at least two interviews separated by several days or weeks, including visualisation. This method was discarded because, since localisation is a comparatively small aspect of their work, developer-generated theories might not be comparable to translator-generated theories. I also felt that participant-formed theories, while interesting and informative, are less conclusive than a theory formed by data. Practical reasons for not choosing this method included an expected difficulty to find participants willing to conduct successive interviews, especially as use of a visual tool complicates remote interviews, e.g. by phone.

Expert interviews aim to obtain views representative for a whole group. I did not use this approach as this implies there is a generalisability of accounts for said whole group, and further a restriction on specific topics. Insofar, there was a concern that this interview style was not open enough. Finally, the research interest was not on expert perspectives, but rather general views.

Ethnographic interviews aim to supplement data from ethnographic observational studies by interviewing participants in an observational study as repeated informal conversations. As I did not conduct an observational study, this interview type was not applicable.

Problem-centred interviews aim to collect biographical data towards a certain problem. This method was ultimately chosen because it focuses on problems and supports interest in subjective viewpoints, facts, social processes, and the aim of developing theory.

Problem-centred interviews are characterised by problem centring, object orientation, and process orientation. It is recommended to start the interviews with a short survey for biographical data so that these do not need to be established during the interview. Problem-centred interviews follow a time-glass model (Runeson and Höst, 2009), where an interview opens with general questions, continues with a phase of specific questions, and concludes with open questions. Problem-centred interviews are recommended to open with a warm-up phase, followed by a phase for general prompting, then specific prompting, and finally *ad-hoc* questions (Flick, 2002). The latter enabled me to engage dynamically with a participant's account and their role, i.e. project manager, software engineer or translator. Overall, the problem-centred interview method combines well with GT, particularly with theoretical sampling and coding (Flick, 2002).

While generally following recommendations, no biographical data survey was conducted prior to the interviews since this was likely to introduce redundancy for the interviewees, who were assumed to have taken part in the online survey. Further, relevant biographical data such as the participant's role or company etc. had already been obtained during interview arrangement and preparation. Unlike the recommendation by Flick (2002), no specific, given problem, was focussed on. Instead, any problem as chosen by the interviewees was discussed. It was also noted that the dynamics of the interview did not always follow the chosen time-glass model, but sometimes had more semblance to the funnel model, with open questions first which then narrowed to more specific questions.

Data collection was first degree, i.e. interviews were conducted by the author. Interviews were recorded provided that interviewees consented. Only minimal notes were taken in order to focus on interview content. The aimed-for interview length was 45 minutes.

During the interviews, recommendations and best practices for semi-structured research interviews from Seaman (1999) and Kvale (2007) were followed. Instructions were kept to a minimum. Interviewees were reminded that participation was voluntary and that they could choose to skip any question for any reason, without giving a reason. It was stressed that there are no right or wrong answers.

It was attempted to steer interviews towards a specific subject initially, but interviewees were allowed to steer the interviews to whatever they found relevant. It was aimed to leave the thematic part of the interview to the interviewees so that I would only participate in the dynamic part, i.e. progressing the interaction. It was attempted to keep questions short, to avoid direct questions inviting speculation, and to only ask for clarification when needed. Instead, interviews were aimed to be self-reported stories with spontaneous information, but little explanation.

Analysis was conducted in parallel with the interviews as described in the GT research process. During the interviews, interviewees were asked to relate their experience in software localisation, their general role, how they are professionally related to software localisation, and what their day-to-day work activities are. Interviewees were encouraged to share anecdotes related to software localisation, e.g. issues they had encountered or witnessed themselves. This included the setup of the localisation process, e.g. whether localisation is done in-house or through an LSP, what the relationship is between members of the development and localisation discipline, and similar, as far as this was known by the interviewees. The overarching interests lay on two complexes: localisation issues and procedural motivations:

The first complex dealt with details of localisation issues encountered by interviewees, i.e. what exactly the problem was, the underlying mechanics leading to the creation or manifestation of the issue, how issues were detected and handled, and what consequences, if any, they had on development and localisation going forward.

The second complex dealt with underlying assumptions and thought processes, if any, having led to the organisational structure, processes, tool usage etc. as encountered in the interviewees' localisation setup.

3.2.3.3 Tools Used During Analysis

After the interview, recordings were transcribed using a simple media player and text editor[40]. Non-English interviews were coded in the original language, only quotations appearing in the write-up having been translated.

The necessity to use tools other than writing utilities during the application of GT was noted already by its original creators. For example, Strauss and Corbin (1998) suggest to conduct sorting with self-sticking notes on a whiteboard or wall. There are a number of software tools available to help with the analysis of qualitative data. I originally started analysis by coding in Weft QDA, an open-source tool for coding text data. After coding a few interviews, a number of technical limitations of Weft QDA such as the inability to edit interview transcripts, led to a switch to NVivo, which I used for most open coding. NVivo is a commercial software package for qualitative data analysis. It serves as a data repository and processing tool and enables researchers to code data and organise, sort, link and arrange information and data. The created data structures can later easily be modified. NVivo further offers search and query facilities to browse and examine existing data, calculate simple coding statistics such as code density, graphically represent links and structures, and supports coding and analysis through multiple users. It supports most computer media, i.e. text, images, movies and sound recordings. However, for this research, imported in NVivo were only interview transcripts and notes.

Towards the end of open coding, I ceased using NVivo because I found that its use had influenced how I understand and process the original data. Specifically, I found my thinking about the data to be shaped not by my ideas of the data itself, but by its presentation, hierarchical organisation and abstraction in NVivo. A similar observation has been made by Hoda *et al.* (2012), who noted a limiting effect on interaction with the data through the use of NVivo. My main worry was that I might begin to rely too much on the suggested ways of handling data inherent in NVivo. Incidentally, I had noticed occasional,

[40] Details of tools are listed in Appendix F.

slight shifts in my codes from the meaning when creating the code, to the meaning associated with it when assigning later items, and I attributed that to the way how existing codes are presented and selected in NVivo.

I eventually switched to using text files for selective coding and sorting. While handling codes, concepts and memos in text files is much more cumbersome, it forced me to re-read and engage more with existing data, for example when adding new codes or reassigning text selections. Having all data in what effectively is a one-dimensional order of which you can only see a limited selection at the same time also made me know my data much better. I would have liked to try sorting with self-sticking notes on a whiteboard as suggested by Strauss and Corbin (1998), but none was available.

3.3 Quantitative Research

RQ3 and RQ4 will be answered through quantitative research. For each research question, appropriate hypotheses were formulated. Table 3-1 lists hypotheses and their relationships to the research questions. Further modifications and additions are discussed during the construction of the survey and choice of statistical tests.

Table 3-1 List of hypotheses

RQ	ID	Hypothesis
3	H1	Developers score lower than localisers on CQ
	H2	Developers assume a different localisation scope than localisers
	H3	Developers score lower on ATL than localisers
	H4	Developers assume less responsibility for localisation than localisers
	H5	Developers have a higher SEL than localisers
	H6	Cost, quality and time priorities differ between developers and localisers
	H7	Software success factor priorities differ between developers and localisers
	H8	Localisation training is correlated with CQ
	H9	Native English speakers score higher than non-native English speakers on CQ
4	H10	LE is affected by software type
	H11	LE is affected by user type
	H12	LE is affected by customer-user identity
	H13	LE is affected by number of target languages
	H14	LE is affected by development model
	H15	LE is affected by project commerciality
	H16	ATL is correlated with CQ
	H17	SEL is correlated with CQ
	H18	SEL is correlated with ATL

3.3.1 Selecting Quantitative Methods

Next, relevant quantitative methods and the method selection to examine the hypotheses about software localisation practice is discussed.

3.3.1.1 Experiments and Quasi-Experiments

Experiments are studies testing causal relationships by applying a treatment to one of two identical situations and observing the effect. It is important that the situation receiving the treatment is chosen randomly; otherwise, the study is referred to as quasi-experiment. Usually, experiments are used to test hypotheses and are repeated many times to be able to make inferences through statistical analysis. It must also be assured that the two situations only differ in the independent variable, i.e. treatment. Therefore, experiments are generally conducted in laboratories.

Experiments have been used in software development research. For example, Shneiderman *et al.* (1977) showed through various experiments that flowcharts are of no help whatsoever to software engineers, neither for debugging nor programming nor maintenance. Tichy (1982) tested the performance of a revision control system through experimentation. Solheim and Rowland (1993) measured effectiveness and efficiency of several integration tools and strategies. The results suggested top-down and big-bang integration strategies.

A properly set-up experiment is considered the ultimate test of a theory, but requires a testable hypothesis. Experiments as such are not necessarily difficult to set up, but most experiments end up having physical requirements that makes experimenting a huge effort. Further, the more complex the context of the situation under examination is, the more difficult it will be to set up an experiment where all influences are controlled.

Experiments can be difficult to recruit participants for as it must be sure that all participants are comparable. Further, it is often argued that the relevance of experimental results is limited due to their laboratory context.

For this research, experimental methods were discarded due to the expected complexity and because no option was apparent how to test the hypotheses within manageable experiments. In principle, software development and localisation could be recreated within a laboratory environment, but the effort to do so seems extreme. In the case of

experiments, it would also require assigning independent variables to the units of analysis. This means randomly educating neutral participants to be either developers or localisers, and simulating software development projects that only differ in user type, number of target languages, etc. Such experiments would have to be repeated many times in order to satisfy the requirements of statistical analysis. Some of these issues could be addressed by conducting quasi-experiments, but this would create new problems, e.g. recruitment of software development professionals willing to spend considerable time for no reward.

3.3.1.2 Survey

Surveys are a way to identify trends or test hypotheses by inference through statistical analysis of samples taken from a population. Survey data can be gathered for example via interview, questionnaire or observation. In computing, surveys have for example been used by Isa *et al.* (2010) to confirm the theory of website information architecture as a five-factored multidimensional product. Blackburn *et al.* (1996) studied software management practices in Western Europe via survey in order to determine management practices supporting higher productivity, and to provide supporting evidence of factors reducing cycle time.

The survey method is comparatively simple to apply. It shares with experiments the difficulty of recruiting participants conforming to the requirements of statistical analysis.

In this research, the survey method was chosen because it was deemed the easiest way to confirm the hypothesis questions while avoiding the construction of experiments. However, survey construction is a complex field with a number of guidelines to consider. Similar to experiments, surveys require a comparatively large number of data points to deliver statistically representative results, which disqualified observation. It was deemed best to avoid redundancy in data collection methods in order to avoid single points of failure. Because interviews were conducted for the GT study, a questionnaire was constructed to realise the survey.

3.3.2 Questionnaire Construction

The survey was designed as a cross-sectional study. The research questions motivating the survey revolve around two units of analysis: for RQ3, it is professionals, and for RQ4,

it is software projects. Because it was considered likely that most respondents have worked on more than one localised project, they were instructed to answer project-related questions for the most recent localised project they had worked on.

To enable statistical analysis, quantifiable data was needed. Accordingly, the survey used multiple choice questions. Text entry fields were only provided as alternatives for questions where it was considered conceivable that the given options would not cover all possible answers.

The hypotheses require measurements of five constructs. These are Attitude Towards Localisation (ATL), Self-Efficacy in Localisation (SEL), Self-Efficacy in Usability (SEU), Cultural Competence (CQ), and Localisation Effort (LE). Further, biographical and opinion data of the participants needed to be collected. Details on the constructs measured and data gathered in the survey are given in the following subsections. The relationship between each question and constructs is shown in Table 3-2. The complete questionnaire can be found in Appendix A.

Table 3-2 Relationship between survey questions and constructs

Q. #	Construct
1 - 5	Biographical data
6 - 24	Attitude Towards Localisation (ATL)
25 - 29	Self-Efficacy in Localisation
30 - 34	Self-Efficacy in Usability
35 - 54	Cultural Competence (CQ)
35 - 38	Metacognitive CQ
39 - 44	Cognitive CQ
45 - 49	Motivational CQ
50 - 54	Behavioural CQ
55 - 60	Project properties
61 - 68	Localisation Effort
69 - 71	Biographical data
72 - 75	Opinions on localisation

3.3.2.1 Biographical Data

The survey collected biographical data on age (Q.1), gender (Q.2), nationality (Q.4), level of education (Q.5), usual role in software development (Q.69), years of experience in software localisation (Q.70), and received training in software localisation (Q.71). These

items were used to characterise the sample. An additional item asked the participant to confirm involvement with the development of international software (Q.3) in order to filter out ineligible participants. Usual role in software development was further used to sort participants into developer or localiser groups. Training in localisation and nationality was used for correlation testing with CQ.

3.3.2.2 Opinions on Localisation

The survey further queried participants' opinions as part of the examination of differences in perceptions and attitudes of developers and localisers regarding localisation scope and project management. Specifically, participants were asked to define their preferred localisation scope from a list of items, to order a number of software quality items and project management items according to their personal perception of priority, and to state whether they feel responsible for localisation. The available options for items to be localised in software (Q.72) was compiled from similar lists in the literature (Anastasiou and Schäler, 2010; Ryan *et al.*, 2009; Collins, 2002; Carey, 1998). The options of priorities in software (Q.73) were inspired by an article of Cook (2011) about different software priorities for scientists and engineers. The project management options of Q.74 were derived from the project management triangle (Dunne, 2011). The question of the participant's personal responsibility of localisation (Q.75) is a straightforward yes/no question.

Because no statistical test could be identified to test for differences in list sorting, it was decided to test the selection chance of each item (H2a to H2g) and the difference of selected item count (H2h) depending on participant role. Because there were only three options with six possible permutations for the project management triangle items, the cost, quality and time priorities could be tested using a Chi-squared test. However, eight software success criteria allow 40320 possible permutations compared to a relatively small number of participants (n = 120), meaning that a different testing method had to be found. Hence, the software success criteria were tested individually with additional hypotheses (H7a to H7h).

3.3.2.3 Attitude Towards Localisation (ATL)

Table 3-3 General changes to ACT

Q. #	Change
11, 13, 17	Replacement of terms, e.g. *computer technology* with *software localisation*
6, 8, 9, 10, 15, 16, 19, 23, 24	Slightly rephrased to account for the differences of the tool *computer technology* versus the concept or process *software localization*
7, 12, 14, 18, 21, 22	Adaptation of a stated use of computer technologies to a purpose for software localization
20	Adaptation of an explicit motivation for an emotional response

Table 3-4 Semantic changes to ACT

Q. #	Class	ACT	ATL	Source
7	use of computer technology/localisation	Communicate with others in order to be more effective on the job	Applying knowledge in software localization to create more effective software for international users	Dohler (1997)
12		Create materials to enhance the performance on the job	Increasing the user base of the software projects one is working on	Kumhyr *et al.* (1994), McKethan and White (2005)
14		Use word-processing software to be more productive	Increase the usability for software one is working on for international users	Aryana and Liem (2011)
18		Access many types of information sources for one's work	Necessity for software projects to adhere to local laws and customs	Ryan *et al.* (2009)
21		Assist in work organization	Avoid misunderstandings and offenses for the software one is working on	Anastasiou and Schäler (2010)
22		Learn new skills	Improve software for international users	-
20	emotional response to technology/localisation	Anxiety towards missing knowledge how to handle errors	Anxiety towards loss of control	-

Table 3-5 Examples of semantic changes to ACT

Q. #	ACT	ATL
9	Using computer technologies in my job will only mean more work for me.	If the software project I am working on is localized, this will only mean more work for me.
20	I am anxious about computers because I don't know what to do if something goes wrong.	I am anxious about software localization in my projects because it might interfere with my efforts or ideas.
21	Computer technologies can be used to assist me in organizing my work.	Software localization helps avoid misunderstandings and offenses for international users of the software I am working on.

The instrument measuring attitude towards software localisation consisted of 19 items, Q.6 to Q.24, which were adapted from an instrument to measure Attitude towards Computer Technology (ACT) that had originally been developed to measure attitudes towards computer technologies of students and education professionals, but had later been adapted for general use (Kinzie *et al.*, 1994). Its 19 items contain both positive and negative phrasings and measured the constructs usefulness and comfort/anxiety with regards to computer technologies. Usage of this questionnaire as a template for our own quantitative research seemed appropriate because of the approach to understand attitude towards computer technology as being made up of the subscales of usefulness as reported by the participant and comfort/anxiety. Both are subjective perceptions and seem appropriate measures of the instrumental nature of software localisation. It was also considered to be suitable since software localisation is considered a particular facet of the broader term computer technology.

Computer technology in the sense in which it was used by the authors of the original questionnaire refers to a concrete use towards an end, e.g. to communicate with others (item 1 in the original ACT), or to learn new skills (item 17 in the original ACT). In that way, it was deemed very similar to software localisation, just with different ends. However, in the original ACT, computer technology is treated as a tool that can be applied directly by the participants. In contrast to that, the adapted questionnaire looks at software localisation as a concept that is part of the participant's work, and at its most concrete might be a process the participant is integrated in. Hence it was not possible to simply

replace each occurrence of *computer technology* with *software localisation* in all items. Instead, four different kinds of changes were made as described in Table 3-3. Some were superficial, others affected the semantics of items. The latter changes are described in detail in Table 3-4 and Table 3-5. The original ACT item ordering was retained in the ATL.

3.3.2.4 Self-Efficacy in Localisation (SEL) and Self-Efficacy in Usability (SEU)

Ten items measured the two constructs Self-Efficacy for Localisation (SEL) (Q.25 to Q.29) and Self-Efficacy for Usability (SEU) (Q.30 to Q.34). In general terms, self-efficacy is a person's perception or confidence in how far it can exert influence in what happens around said person, or towards a specific subject or task (Agarwal and Karahanna, 2000; Bandura, 1977). The measurement of SEU was created as a control measurement to control that a specific score for SEL is not part of a general self-efficacy trend. Usability was considered to be a good subject for this because, similar to software localisation, it is a concept without immediate application. In comparison, user-centred design is a well-defined method, and usability testing is a specific process. Both are related to the concept of usability and are procedural manifestations of the concept, without explaining the concept in full. Two additional hypotheses were created: H5a states that localisers score lower on SEL than developers, and H5b states that SEL is correlated with SEU.

New items measuring self-efficacy for software localisation and self-efficacy for usability were created based on the self-efficacy for computer technologies test by Kinzie *et al.* (1994). For each construct, five common work steps associated with software localisation and usability were identified and the confidence of developers to do these was queried. For software localisation, these were applying localisation functionalities of UI frameworks (Q.25), gathering context information for translators (Q.26), identifying software elements to be internationalised (Q.27), handling translated or localised content (Q.28), and using Unicode (Q.29). For usability, these were creating a clear UI (Q.30), conducting usability tests (Q.31), formulating error messages (Q.32), analysing usability test results (Q.33), and implementing changes suggested by UI experts (Q.34).

3.3.2.5 Cultural Intelligence (CQ)

An existing validated instrument was chosen to obtain Cultural Competence in Q.35 to Q.54. What was needed was a measurement of cultural competence relevance to

software localisation, i.e. what aspects in software need to be adapted for various locales. The principal appropriateness of cultural assessment in the context of software engineering to improve product localisation was noted by Linna and Jaakkola (2010).

A wide range of cultural assessment tools for selection and development purposes are available[41], and a number of tools were reviewed for use in this research. Matsumoto *et al.* (2001) developed the Intercultural Adjustment Potential Scale (ICAPS), a tool predicting cultural adjustment, but this was specifically aimed at Japanese sojourners to predict their working success in foreign locales and thus not applicable. Albir and Alves (2009) and Malmkjaer (2008) discuss the concept of Translation Competence, but the competence to translate or localise something goes beyond what is expected of developers – that's why the roles of localisers and developers exist in the first place. For a time, it was considered to use the test of Thomas *et al.* (2012) to measure cultural intelligence as a construct consisting of knowledge, skills and metacognition, i.e. a self-reflective aspect. Ultimately, the Cultural Intelligence Scale (CQS) by Ang *et al.* (2007) was adopted because it fits with the research purpose, is easy to apply, and is readily available. Further, the inclusion of motivational aspects were considered to be an advantage, and not a distraction, as suggested by Thomas *et al.* (2012). Ang *et al.*'s interpretation of the concept of cultural competence seems close to the understanding relevant for software localisation, particularly because it is further broken down into four sub-constructs and is designed to handle "an individual's ability to grasp and reason correctly in situations characterized by cultural diversity" (Ang and Van Dyne, 2008, p.4).

With the CQS, Ang *et al.* (2007) aimed to create an instrument to measure a construct they called Cultural Intelligence (CQ), described as the "capability to function effectively in culturally diverse settings […] arising from differences in race, ethnicity and nationality" (Ang *et al.*, 2007, p.335). CQ is supposed to predict cultural judgment and decision making as a cognitive process, cultural adaptation in terms of sociocultural adaptation and well-being, and task performance as the conduct of prescribed activities. Although the test is aimed to predict functioning within cultural context foreign to the subject, Ang *et al.* understand this as a cognitive process, requiring conscious understanding, perception and processing of cultural differences. Hence, it was considered that Ang *et al.*'s definition of

[41] A comprehensive list is given in Linna and Jaakkola (2010).

CQ would also be relevant in the context of cultural competence for internationalisation and localisation.

Ang *et al.* measure CQ through four sub-constructs: Metacognitive CQ refers to mental processes to acquire and understand cultural knowledge, describing the ability to understand one's own cultural preferences and the cultural preferences of others. Cognitive CQ refers to knowledge of other cultures' preferences such as conventions and customs. Motivational CQ means intrinsic interest in learning about other cultures. Behavioural CQ is the ability to exhibit appropriate cross-cultural behaviour, respectively control one's own behaviour accordingly.

Ang *et al.* developed the CQS to be unaffected by the cultural background of individuals taking the test, and to be reliable, valid, and stable over time. For this research, the instrument was adopted without changes so that validity and reliability are retained. It is expected that behavioural CQ and motivational CQ are not as relevant to software localisation, whereas cognitive CQ and metacognitive CQ, i.e. the actual knowledge of other cultures and the ability to think about other, unknown cultures in an abstract way, for example in order to anticipate reactions, are obviously relevant for successful internationalisation and localisation.

In order to test the sub-constructs metacognitive CQ, cognitive CQ, motivational CQ and behavioural CQ as well, hypotheses were added to applicable tests on CQ, e.g. H1 "Developers score lower on CQ than localisers" was followed by H1a "Developers score lower on metacognitive CQ than localisers", H1b "Developers score lower on cognitive CQ than localisers", and so on. It was further decided to test the correlation of cultural competence, attitude towards localisation and self-efficacy in localisation, creating additional hypotheses (H16 to H18).

3.3.2.6 Project Properties

Project properties were surveyed in order to answer RQ4. The respective hypotheses test whether there is a correlation between given project properties and localisation effort. The project properties are software type, user type, whether users and customers are identical, number of target locales, development methodology, and project commerciality.

The options for software type (Q.55), user type (Q.56) and development models (Q.60) were generated from my own understanding of what would be clear and unambiguous options covering the subject area as wide as possible, but influenced by respective discussions in Ryan *et al.* (2009). Options for customer-user identity (Q.57) and project commerciality (Q.58) were straightforward confirmations or rejections, where customer-user identity also allowed a mixed option.

Rather than asking for target locales, the survey question asked for target languages (Q.59) to avoid confusing participants unfamiliar with the term locale. It was assumed that in this context, number of locales and number of languages would be identical anyway. Participants were asked to choose from a range of options rather than input an integer into a text field because not all participants were assumed to actually know the exact number of target languages. In order to minimise guessing errors, the answer options were designed broadly and the number of options was limited to four. The upper distinction limit originated from a general impression based on the literature that few software projects localise into more than 30 languages. The rationale behind the thresholds, 5 and 15 languages respectively, was the assumption that participants are more likely to know the number of target languages when this number is small, i.e. the fewer target languages there are, the more likely participants are to take note of or remember.

As applicable, options to specify unknown or other values were added. For example, as answer options for number of languages covered everything from 1 to infinity, no other option was applicable, and software type had no unknown option as participants should have a general idea what they are working on.

3.3.2.7 Localisation Effort (LE)

The construct LE was operationalised based on efforts, i.e. activities, processes or tools, identified in the literature to lower localisation cost, increase localisation time, and shorten localisation duration. Origins and support for each item (Q.61 – Q.68) are identified in Table 3-6. The items were further selected to have a reasonable chance to be known by participants from both the developer and localiser group.

Table 3-6 Origins of LE items

Q. #	Localisation Effort (LE) item	Source
61	Clear localisation requirements	Law (2003)
61	Best practice guidelines	Giammarresi (2011)
61	Glossary	Giammarresi (2011)
61	Translation storage and re-use	Freigang and Reinke (2005), Bowker (2005)
61	Dedicated localisation engineers	Giammarresi (2011)
61	Possibility for all developers to compile localised versions	author's experience
61	Simshipping	Ryan *et al.* (2009), Hartley (2009), Kahler (2000)
62	Emphasis of localisation quality	Law (2003)
63	Localisation scope	Cyr and Trevor-Smith (2004), Ryan *et al.* (2009)
64	Translation source	Morado Vázquez and Mooney (2010)
65	Localiser communication	Law (2003), Collins (2001), Russo and Boor (1993)
66	Localisation file format	Law (2003), Sachse (2005), author's experience
67	Context information	DePalma (2006), Honkela *et al.* (1997)
68	Quality assurance efforts	author's experience

Q.61 queried a number of nominal scale items, with each selected item contributing to overall LE. Q.62 to Q.68 offered options that were considered on an ordinal Guttman scale (Guttman, 1974) so that the more laborious an option is, the higher a score it contributes to overall LE. For example, Q.64 asked where the translations for the project originated, with MT assumed to indicate low LE and hence counting least, and full-time employees assumed to indicate high localisation effort and hence counting most. Each individual LE item was tested separately (see H10a to H10f and H11a to H11f).

3.3.3 Survey Presentation and Pilot

The survey was implemented as website using the questionnaire software LimeSurvey[42]. It was structured in seven web pages:

1. Introduction, informed consent and instructions
2. Part 1: Biographical data
3. Part 2: ATL, SEL, SEU
4. Part 3: CQ
5. Part 4: Project properties

[42] Details of tools are listed in Appendix F.

6. Part 5: Additional biographical data, opinions on localisation
7. Opportunity for feedback and registering for results

The questionnaire was piloted with ten individuals considered to be typical study participants, i.e. software engineers, project managers and translators. The pilot study was conducted in order to check for ambiguity in wording, unexpectedly high response of the default option, and any other potential problems. In addition, pilot participants were asked to time how long it took them to complete the questionnaire so that a good estimate for potential participants could be provided.

The pilot participants did not report any significant issues with the questionnaire so that the only changes were corrections of typos and odd wording. The average completion time was ca. 15 minutes.

3.3.4 Survey Analysis

The constructs and scales used for the analysis are listed in Table 3-7. Statistical analysis was conducted using the Statistical Package for the Social Sciences (SPSS)[43]. SPSS is a statistical analysis software allowing researchers to calculate descriptive, analytical and predictive statistics. SPSS handles data in the form of tables, with rows representing cases and columns representing variables. Common statistical measures have been implemented and can be applied to the data. SPSS further features extensive data management features for variable naming and scaling, outlier handling and data filtering, and offers a number of options to output descriptive and analytical results as tables, charts or graphs.

There are no interventions and the survey is not a longitudinal study. The statistical analysis therefore has three different objectives:

The first objective is to test whether averages, nominal variables and ordinal variables between two groups, e.g. developers and localisers, differ. To test differences of averages, an independent-samples t-test or a Pearson correlation test is used. To test differences of ordinal and nominal variables, the Phi coefficient correlation test is used. To test differences of groups of variables, the Chi-square test is used. The second

[43] Details of tools are listed in Appendix F.

objective is to test whether averages between multiple groups, e.g. software users, differ. To test this, an analysis of variance test (ANOVA) is used. The third objective is to test whether two variables are correlated. To test correlation of two ordinal variables, the Spearman rank correlation test is used. For all tests, the SPSS standard implementation is used. All used tests fall under Fukuda and Ohashi (1997) standard tests.

Table 3-7 Scales of constructs

Construct	Scale
Cultural Competence	ordinal
Attitude Towards Localisation	ordinal
Self-Efficacy in Localisation	ordinal
Self-Efficacy in Usability	ordinal
Nationality	nominal
Role in Software Development	nominal
User Type	nominal
Software Type	nominal
Customer-User Identity	nominal
Project Commerciality	nominal
Number of Target Languages	ordinal
Development Methodology	nominal
Localisation Effort	ordinal

The survey unit and the analysis unit for parts 1, 2, 3 and 5 of the questionnaire was the participant. For part 4 of the questionnaire, the analysis unit was most recent project the participant had worked on. Participants were categorised into two groups: software engineers, UI designers, project managers working on engineering projects, and other participants with a focus on work in and around software engineering or in software companies were categorised as developers. Translators, technical writers, or other personnel working on localisation or in translation companies were categorised as localisers.

3.4 Population and Sample

Considering what kind of sampling method is used is an important aspect for quantitative research because it has a strong influence on generalisability. A sample is either a *probability sample*, where each element of the population under examination has a known probability to be selected, or a *nonprobability sample*, where at least some

elements of the population have an unknown or no probability to be selected. The advantage of a probability sample is that the sampling error is known, which ultimately allows for inductive inferences about the entire population. Nonprobability samples, on the other hand, must be considered non-random samples without information about the sampling error. Hence the representativeness of the sample for the entire population is limited or dependent on assumptions made during sampling.

Target participants for the study were professionals with experience in contributing to international software. Respondents were expected to have actively contributed to such a software project, for example in the role of software engineer, translator, UI designer or project manager, in line with localisation contributors described in the literature (e.g. Hartley, 2009). By leaving the target group relatively unrestricted, it was hoped that as many points of view as possible could be examined. For example, it is conceivable that a project manager has different views on the importance of a localisation project than a software engineer, or a UI designer might have more knowledge about cultural differences than a system analyst. However, only professionals were recruited, i.e. people who in principle contribute to software for a living. For example, a marketing staff member is not assumed to directly contribute to the development of a software product.

Requests for participation were published on different media related to software development and localisation:

- 13 mailing lists
- Internal communications (e.g. newsletters) of six topical organisations
- 17 Facebook groups
- 16 newsgroups and internet forums
- 16 twitter targets (accounts and hash tags)
- Two print magazines (one non-topical)

Additionally, study participation was further promoted in person at three industry fairs, four research conferences, and five workshops. More than 200 individuals were mailed individually, mostly as follow-up to responses from the listed recruitment activities.

The applied sampling method has to be considered *convenience sampling*. Convenience sampling, also referred to as *accidental sampling* or *opportunity sampling*, usually refers

to a sample chosen for relatively easy access. However, it is a non-probabilistic sample. Therefore, the sample might contain unknown biases and care must be taken before generalisation. For example, it is conceivable that developers with a positive attitude towards localisation or with an increased cultural competence were more likely to participate in the survey than those with a negative attitude towards localisation. In that case, the test for correlation between localisation role and attitude towards localisation or cultural competence might have been foiled by the bias.

Despite the disadvantages of convenience sampling over other sampling methods, especially with regards to generalisability of survey results, it was nonetheless considered useful. Convenience sampling is suitable for qualitative data, in particular when comments are sought on rich and meaningful topical statements. With the previously discussed restraints on data gathering, it provided a realistic and adequate sampling method.

No attempt was made at avoiding multiple sampling because I could not conceive of a way to do so reliably and without in some way involving the identity of the participant. Early on it was considered to introduce some kind of unique identifier constructed by participants themselves from their date of birth, name and company name. This would have added the possibility of noticing when two survey participations referred to the same company. However, eventually neither accidental nor intentional contamination of survey data through repeated participation seemed probable or plausible. Subsequently ensuring participants' anonymity was preferred to involving their identity and to risk affecting answers or discouraging participation.

Obviously these considerations apply only to the survey. In the interviews, multiple sampling would have been apparent. Further, the sampling considerations in this section applied to the interviews only during the initial phase of the GT research process, i.e. during open coding and axial coding. In the later phase, theoretical sampling was applied, which means that some participants were actively sought out.

3.5 Ethics

The research was conducted based on the ethics codes of the UWL Faculty of Professional Studies (UWL, 2008), the code of ethics of the Association for Computer Machinery (ACM)

(Anderson *et al.*, 1992), and the ethics code of the British Sociological Association (BSA) (BSA, 2002).

All participants were formally briefed about research purpose and methods, their right to anonymity and the possibility to discontinue participation at any time. Participants were informed of any data gathered. In particular, interview recordings were only made with the participant's expressed consent[44]. Participants were treated fairly, honestly and with respect. If desired by the participants, they receive a report on the research findings.

Care was taken to handle specifically cultural aspects sensitively and to avoid open, hidden or unintended racism, offenses and cultural discrimination against. Any data gathered during this research was treated confidentially. This includes data gathered through case studies or other research methods in and from companies, including those acquired from its employees.

3.6 Summary

In this chapter, two research approaches for the research objectives were developed: a GT approach fed from interviews will empirically examine the practice of software localisation in order to understand how developers and localisers collaborate and influence each other, and how localisation issues are caused in this collaboration. An online survey is conducted to determine differences between developers and localisers, the role of cultural competence, and connections between product properties and localisation. The next chapter will introduce the results.

[44] The consent sheet is shown in Appendix B.

Chapter 4 Qualitative Results

To examine how localisation is conducted in practice, what shapes this practice, and how issues are caused during this practice, interviews with developers and localisers were conducted and analysed using GT. This chapter discusses the results of this analysis. First, the interview population is reviewed. Then, it is described how the GT process played out during this research, including an overview of the evolution and emergence of the core and the resulting theory. Finally, the chapter presents a grounded theory of interdisciplinary collaboration in software localisation which describes collaboration strategies and conflicts as a reaction to external constraints, which in turn influence each other. Following recommendations by Wisker (2008), excerpts of the interview data on which this theory is founded are presented and discussed in the light of both existing literature and theories, and ramifications for this research. In following with GT practice, (Strauss and Corbin, 1998; Hoda *et al.*, 2011), this includes theories of a wider scope such as software development, collaboration, sociology and organisational psychology which were not previously discussed in the literature review.

4.1 The Research Process

Section 3.2 described GT and the rationale to choose it as research method for this research, i.e. to understand software localisation as a socio-technical situation for which no suitable theory or framework is known yet. In this section, the participants and the process of interviewing is elaborated upon. Further, the progress of coding and emergence of the core is illustrated and deviations from the formal GT process are identified.

4.1.1 Participants and Interviewing

28 interviews averaging 58 minutes were conducted in total, with the shortest interview being 31 minutes, and the longest 2 hours 17 minutes. Table 4-1 gives an overview. Interviewee nationalities were well mixed. 6 Interviewees were female and 22 were male.

The interviews focused on work activities both conducted and observed by interviewees, the scope of localisation within their projects, their knowledge and educational background, and most prominently their experiences and encountered issues.

Table 4-1 Interviewees

Id	Classification	Role	Date	Length	Lang.	Notes
D1	Developer 1	Development team leader, communication	9 March 2011	00:53	Eng.	
D2	Developer 2	Software engineer, software company	11 April 2011	02:17	Eng.	
D3	Developer 3	Software engineer, web service provider	17 September 2011	n/a	Eng.	no recording
D4	Developer 4	Software engineer, business software company	14 May 2012	00:45	Eng.	
D5	Developer 5	UI designer, business software company	15 May 2012	00:37	Eng.	
D6	Developer 6	Interaction designer, business software company	27 April 2012	00:49	Eng.	
D7	Developer 7	UX designer, software company	22 May 2012	00:47	Eng.	
D8	Developer 8	UX consultant, software company	21 May 2012	00:46	Eng.	
D9	Developer 9	Software engineer, software company	1 June 2012	00:44	Eng.	
D10	Developer 10	Usability architect, software company	7 June 2012	00:49	Eng.	
D11	Developer 11	Software engineer, business software company	12 June 2012	00:38	Eng.	
DM1	Development Manager 1	Project manager, business software company	16 May 2011	00:34	Eng.	
DM2	Development Manager 2	Project manager, financial institution	6 July 2011	00:38	Eng.	
DM3	Development Manager 3	Project manager, software company	12 December 2011	00:31	Eng.	via phone
DM4	Development Manager 4	Project manager, software company	9 January 2012	00:52	Eng.	via phone
DM5	Development Manager 5	Project manager, business software company	10 May 2012	00:45	Eng.	
DM6	Development Manager 6	Software engineer, software company	14 November 2011	01:01	Ger.	via phone
DM7	Development Manager 7	Localisation engineer, software company	22 November 2012	n/a	Ger.	no recording
L1	Localiser 1	Freelance translator	1 March 2012	02:17	Ger.	via phone
L2	Localiser 2	Freelance translator	20 February 2014	n/a	Eng.	no recording
LM1	Localisation Manager 1	Project manager, LSP	16 November 2011	00:42	Ger.	via phone
LM2	Localisation Manager 2	Localisation project manager, software company	17 November 2011	01:15	Ger.	via phone
LM3	Localisation Manager 3	Freelance localisation consultant	2 April 2012	01:00	Eng.	v. ph., part. rec.
LM4	Localisation Manager 4	Business manager, LSP	14 October 2012	01:10	Ger.	via phone
LM5	Localisation Manager 5	Freelance localisation consultant	11 October 2012	00:48	Ger.	via phone
LM6	Localisation Manager 6	Project manager, LSP	3 December 2013	n/a	Eng.	no recording
LM7	Localisation Manager 7	Localisation project manager, software company	30 March 2012	01:10	Ger.	via phone

In order to contrast accounts from different disciplines and roles, interviewees were grouped as developers and localisers. Within each group, it was further distinguished between management and non-management roles. The eventual possible permutations were developers, development managers, localisers, and localisation managers.

Developers are mostly conducting hands-on software engineering activities, e.g. writing program code, designing software architecture, creating user interface layouts, etc. Localisers are primarily translating content, or creating content in foreign locales. Development managers and localisation managers differ insofar as they have to conduct management, i.e. forecasting, planning, organising, commanding, coordinating and controlling (e.g. Bocij *et al.*, 2008), of activities of other developers and localisers.

The groups were assigned after the interviews, based on the job description given by the interviewees and a description of their tasks, but for consistency without their direct input or confirmation.

In all cases, the management role seems to have been inclusive, i.e. managers also engage in development and translation in addition to their management activities. It appears that no members of the development group had any professional or educational linguistic background. On the other hand, some members of the localisation group had a professional programming background.

Unless otherwise noted, the interviews were conducted face to face and recorded with a dedicated recording device[45]. Phone interviews were conducted with voice-over-IP (VoIP) software and recorded with a recording plug-in. With two exceptions, all interviewees agreed to be recorded. In one instance, the dedicated recording device failed from the start. In one instance, the recording plug-in failed halfway through the interview.

Sampling was limited by the volunteers who answered the call for participation, but covered the subject of software localisation sufficiently to assume saturation once the interviews ceded to provide new insights.

Noticeably, the majority of interviewees in the localisation group chose to be interviewed at work and via phone, whereas many interviewees in the development group chose their

[45] Details of all tools used are listed in Appendix F.

spare time to meet in person. I suspect this is because the latter were often recruited at face-to-face events, which already indicates geographical proximity. The former, on the other hand, were recruited via the internet or other indirect means.

It would have been interesting to interview more members of the localisation group face-to-face to see if the opportunity to e.g. sketch on a piece of paper would have made a difference. Further, it would have been interesting to speak to more translators, but although the effort was made to recruit more, either I did not find the right channels, or translators are more reluctant to participate in such research.

4.1.2 Core Emergence and Implementation of the GT Process

Most interviews were conducted during 20 months from early 2011 until late 2012, with two interviews for clarification of individual points late in 2013 and early in 2014. Interviews were conducted following the GT process described in subsection 3.2.2.1. Although incoming survey data was reviewed irregularly to estimate data collection process and survey acceptance, it is not considered an additional data source since no statistical analysis took place until mid-2013, when most interviews had been completed.

It can be argued that I might have had too much experience with and knowledge of software localisation to actually apply GT as intended and could not possibly have come to the research process without preconceptions. During data collection and interpretation, I reflected regularly on the influence my experience might have. Hence, my professional experience should be categorised as theoretical sensitivity as discussed in subsection 3.2.2.1, rather than existing theory as discussed in subsection 3.2.2.3.

Sampling and coding followed the GT research process described in subsection 3.2.2. An interview excerpt with a sample of high-level codes derived from open coding is given in Appendix C. In the excerpt, a participant in the role of project manager describes how he obtains translations for his software product from the customer, and describes the rationale behind the process and how it might be applied, e.g. if translations for new features were necessary. Among others, the sample of high-level codes in the excerpt code training as the source of knowledge about localisation processes and infrastructure, and further code the explication of a localisation process, specifically localisation by customers. Also, the deferral of responsibility to the customer is coded. All these codes

apply to only short sections of text, i.e. between one or more words and a few lines. Another code not explicit in the excerpt relates to the entire excerpt: the replacement of localisation quality with customer involvement. As these codes were derived from open coding, they are more or less exclusive to this interview. Yet another code not explicit relates to the entire interview: the reported localisation scope. This last one is a code shared with virtually all other interviews, as localisation scope is an almost universal metric in this context.

In GT, a theory is derived by developing and evolving the codes across interviews. For a better understanding, a hierarchy or generalisation of codes can be developed, as happened here with the coding of localisation process and localisation through the client, which is a specification of the localisation process. Further, new codes can be retroactively applied to old interviews. In this instance, the code for responsibility deferral prompted a review of older occurrences where responsibility had been deferred by a developer to the customer or other entities. This eventually developed into the task focus strategy featured in the final model. Likewise, the code for reported localisation scope originated from earlier interviews. It allowed to code for variability by comparing across different interviews, here by comparing the localisation scope reported in the excerpt in Appendix C to that of other interviews, for example the excerpt in Appendix D. This prompted the insight that a wider localisation scope, i.e. a more general internationalisation, requires more coding work during localisation.

Figure 4-1 shows the hierarchical node structure that developed during initial open coding, in which of course all the previously mentioned codes or their generalisations can be found, e.g. localisation scope, localisation process, and responsibility assignment. Following a number of interviews, the core, i.e. the central concern in the interviews, appeared to be emerging as issues, more specifically an aggregate of bugs, delays and procedural problems. While the initial node structure had organised localisation issues according to type, the interviewees seemed to be more concerned about the apparent causes. A comparison of issue causes led to the categories internal transgressions, external constraints, institutional bottlenecks and developer-localiser gap. Figure 4-2 visualises these descriptive categories. Further interviews and selective coding and, as far

as possible, theoretical sampling did not contribute any new insights to the descriptive categories, suggesting that saturation had set in.

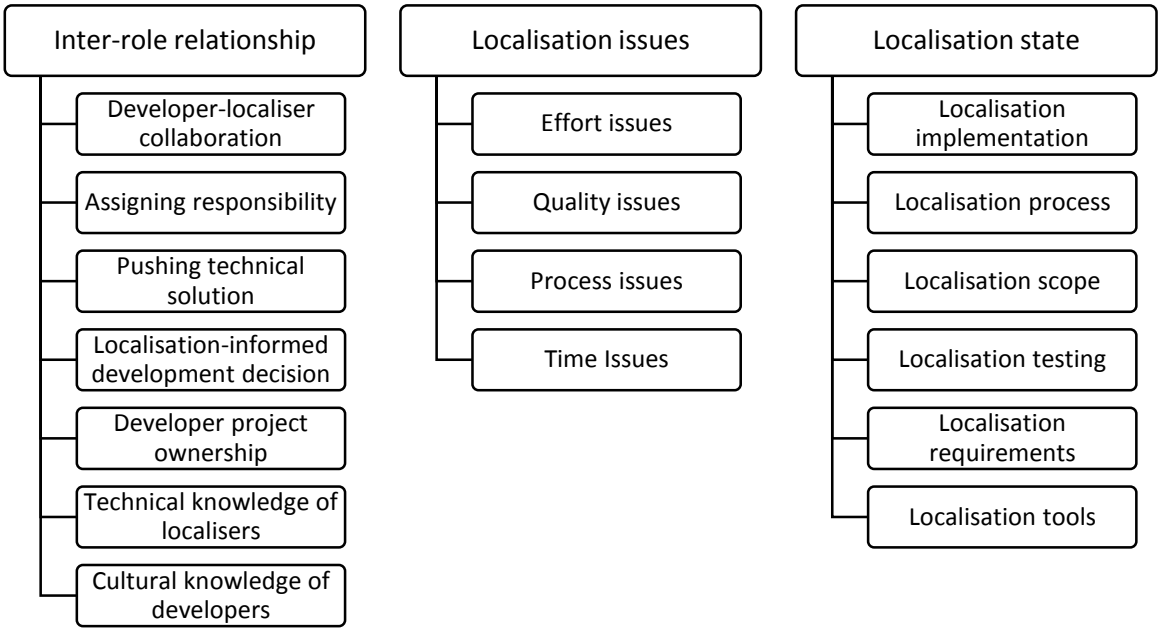

Figure 4-1 Initial coding node structure

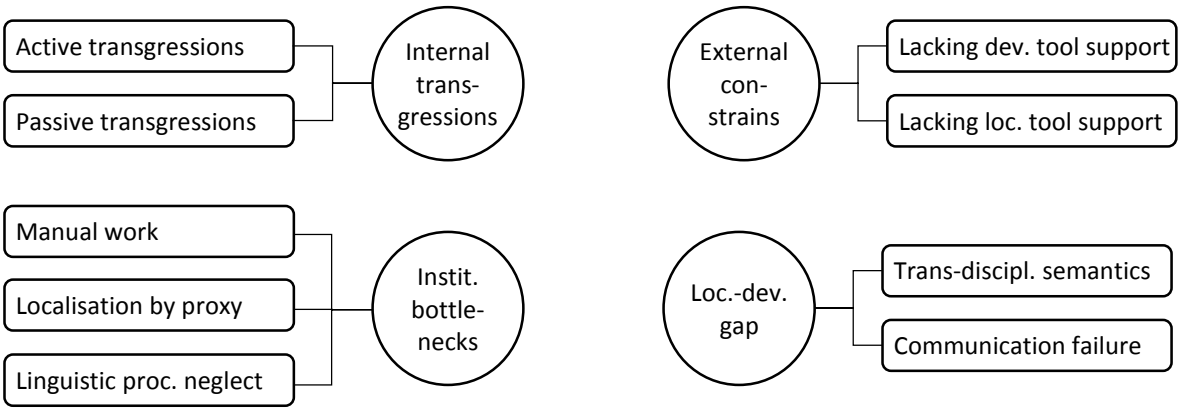

Figure 4-2 Descriptive categories of software localisation issues

Strauss and Corbin (1998) defined a number of criteria for grounded theories. A theory is empirically grounded if it includes systematically related, well developed concepts leading to conceptually dense categories with many dimensional properties. The theory should allow for explained variation and lead to findings potentially significant outside of the original context.

Based on these criteria, the descriptive categories of software localisation issues were not satisfactory in three regards. First, the core, i.e. the aggregate of issues, confounds a number of different topics into a vague and undefined central theme. Second, the relationships between the concepts and categories were considered thin and the entire result primarily descriptive. And third, the concepts and categories are very specific to software localisation and thus less significant in other contexts.

This realisation prompted a revisit of the data. Re-reading the interview transcripts, it became apparent that the core of many accounts were not actually localisation issues. Instead, the interviewees' main concern was how they facilitated and perceived their interdisciplinary collaboration regarding software localisation. Because this insight came from the existing data, it was decided to sort and categorise it again without conducting more interviews.

The importance of external influences and strategies was already part of three of the four descriptive categories. In order to avoid another descriptive categorisation, existing codes were examined for variability and properties. Further categorisation based on similarities were avoided, and instead sequential activities or implied causalities were considered. Directness of contact was identified as dimension. Eventually, the reordered and new categories led to a grounded theory of interdisciplinary collaboration in software localisation described in the next section.

4.2 A Theory of Interdisciplinary Collaboration in Software Localisation

In this research, it was found that for the interviewees of this research, their work in software localisation centres around the *facilitation of interdisciplinary collaboration*, that is, the main concern identified in our research is the interviewees' focus on conducting their work while directly or indirectly collaborating with members of another discipline. The work is critically directed by *external influences on software localisation*, i.e. those conditions, limitations and parts of the environment which the interviewees cannot affect. These influences also affect the choice of *strategies to facilitate interdisciplinary collaboration* employed by interviewees. Finally, interviewees experienced *conflicts of interdisciplinary collaboration*, which have their origins in part in the chosen strategies and the external influences.

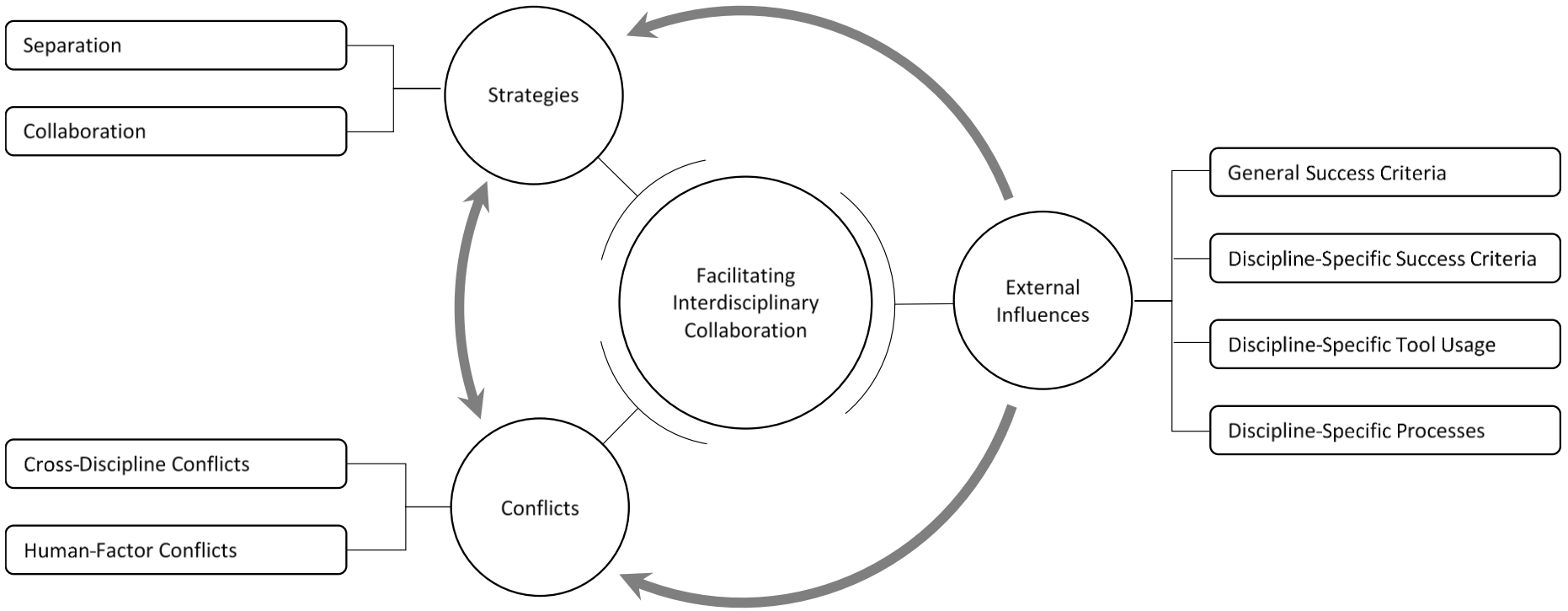

Figure 4-3 Emergence of interdisciplinary collaboration during software localisation

During the research, the grounded theory emerged *bottom-up*, that is, from interview data which was processed, reduced and organised. The theory is presented here *top-down* as a chain of evidence. Figure 4-3 gives an overview of the grounded theory of interdisciplinary collaboration, with circles signifying categories, boxes signifying high-level concepts, lines indicating relationships, and arrows indicating influences. Following, each of the categories, external influences, conflicts and strategies is described in detail, including concepts and narratives from which they emerged.

4.2.1 External Influences

The work of software developers and localisers is influenced by factors outside of their control. These are referred to here as external influences, which does not necessarily mean that they are external to the organisation or business unit developers and localisers work in. Figure 4-4 gives an overview of the specific concepts in this category.

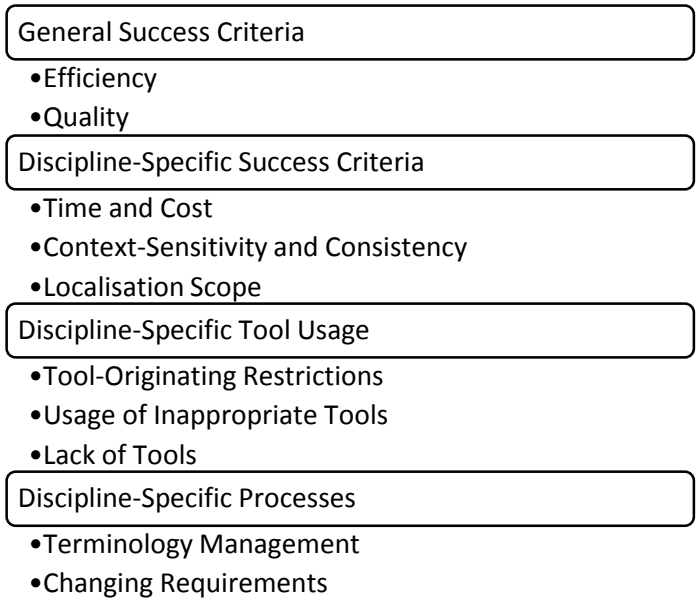

Figure 4-4 Emergence of the category External Influences

The importance of external influences lies in their impact on strategies and potential to create conflicts. For example, cost can control processes in interdisciplinary collaboration:

> *LM7: Between the first version and following [versions], new texts are developed and written all the time. For these, new translations would be needed continuously. So, you need to point out to the software development department, "You cannot just send two or three texts to the translation agency. You might be able to do it once, and because they try to please you, they translate it, in 17, 18, 19 languages. But you cannot do it continuously." […] A translation agency has overhead*

handling work as well, so you have minimax prices [n.b.: minimum charges]. And when I tell the [development department], "Sure, I can have two texts translated, but that's 400 €", they ask, "Well, why?" That's always a problem, you need to collect new texts and wait until there are 20, 30, 40 texts, until it is effective to send it to translation. Of course you have to communicate that internally, and of course there are difficulties with that.

Author: Was that a problem, to communicate that? Or was the problem solved eventually? […]

LM7: This happened regularly, yes, of course. So, specifically things like, "Ok, now, we need to put in this new function, we need five texts for it." And that was no problem. But [the translation agencies] eventually sat us down and said, "Sure, we can translate that for you, that's 400 €." And the [development] project leader had to decide: Is it worth it, or is it not? So that was a clear judgment call. In the beginning it is difficult, but after you do it a few times, it catches on with the project leader. So they need to judge it based on their budget and say, "Ok, that's worth it, we want this functionality in as fast as possible, it must be in tomorrow, please have these five texts translated for that price", or they said, "Never mind, it is not that urgent, next week more [texts] will be added, and then we will do it." […] Afterwards, the awareness was there that this is simply a process with costs that takes time.

Another example how the influences described here impact strategies is illustrated by the relationship between context inquiries and remuneration for translators.

Some accounts suggest that lack of tools or inappropriate tools can lead to conflicts and eventually affect strategies by forcing manual work or transgressions. Likewise, discipline-specific processes have been shown to lead to activity conflicts.

The impact of external influences, such as standards, practices, and even the eventual user, on translation is well known. It is not restricted to translation as a product, but extends to translators' activities. The initiator as well as processes preceding translation play a vital role here (Moorkens, 2012a), and this fits well to the category of external influences. Schubert (2009, p.17) has discussed the "controlling influence" from outside on the work of technical translators, and lists many of the influencing factors found in this research, included document management processes, job specifications, initiators and consumers, source quality, standards, and even best practices of collaborating disciplines. Combe (2011) implies influence categorises processes, and tools including programing languages and authoring tools.

In software development, conditions set outside of the development organisation similarly determine activities of developers and outcome of development (Quintas, 1993; Grinter, 1996a, pp.115, 188). Ferreira (2011, p.200) noted that collaboration or separation of developers and designers in the context of HCI "depend on the values endorsed by the organisations in which the developers and designers are embedded." Maybe the increased dependency between interdisciplinary collaboration and the environment, respectively the conditions, is that the latter do not only affect a discipline's work in itself, but also mediate the influence of the collaborating discipline. Within the sociology of work and work psychology, the influence of the environment and external conditions is of course acknowledged. Work, conditions of work, activities and results continuously affect each other. Thus, changes in any single item, e.g. in the conditions, immediately affect activities and work results, and particularly the subjective experience of any conditions, changes etc. by the worker. Insofar, attempts to control work by setting conditions are conditional to the workers' subjective experience of them (Baron, 1995; Hacker, 1986). This seems to be true particularly for interdisciplinary work.

Analysis of the interviews led to four categories of external influences: General and discipline-specific success criteria, tools and processes. These are illustrated and discussed next.

4.2.1.1 General Success Criteria

The interpretation of general success criteria by each discipline shapes their professionals' activities. Success criteria for a software product are an important factor in shaping the work of both software developers and localisers. They are general insofar as they can be observed in both disciplines, but each though each discipline might interpret and prioritise it differently, or try to achieve it in different ways. In any way, general success criteria drive activities within a discipline and thus affect both localisation process and outcome.

4.2.1.1.1 Efficiency

Efficiency refers to achieving the most possible benefit at the least possible cost or effort. The pressure to minimise cost is felt throughout all aspects of software development.

Cost minimisation influences recruitment choices, processes and activities. And cost in particular can apparently serve as powerful corrective.

For an LSP, on the other hand, efficiency includes planning translation processes for multiple projects and keep the pipelines for translators and revisers full:

> *The most important of all is planning the translation/revision process in a way that there is no way to miss a deadline, both translator and reviser have enough time to do their best but also not to engage them too long so as they will be ready for new assignments as soon as they are finished. (LM6)*

LM6 goes on to emphasize that it is vital for translation agencies to build a good relationship with its freelancers, that is, to always provide them with enough work so that they will be constantly available. This drive for efficiency is also noted among translators, who are very conscious of the relationship between the time they spent on a particular piece of translation and their hourly rate.

4.2.1.1.2 Quality

In localisation projects, there are different perceptions of quality, and accordingly, quality as criterion guides software localisation in different ways.

Among software developers, quality is predominantly understood as a part of usability. As DM3 puts it: "You screw localisation, you screw usability. As simple as that." DM5 had also equated localisation with usability and accordingly felt that it was appropriate to apply usability principles, e.g. consistency across locales. Similarly, DM5 named Hofstede's cultural model for guidance on locale differences. Clearly, localisation and usability are related. However, there might be a danger that this view limits the understanding of localisation, which should be a bit more comprehensive, e.g. considering the question of acceptability. It might be that developers prefer considering localisation quality in terms of usability due to a lack of metric for acceptability. LM5 noted the vagueness of localisation requirement definitions:

> *They say, [localisation] has to be adequate, and that's it. But there is nothing about what criteria exactly to apply. (LM5)*

There seems to be little in the way of concrete quality criteria. As LM5 added later, unlike localisation quality, time and particularly cost requirements are well defined. Further, in

his experience, if localisation is considered a technical software aspect without specific requirements, it just will not be made properly. DM3 made a similar observation:

> *Developers couldn't care less about financial regime or financial processes that goes in a given country. […] [Localisers] have to be able to translate it into functional requirements that are understood by the developers. (DM3)*

Localisers' idea of localisation quality is different, aiming more for linguistic quality and cultural appropriateness. For example, LM 4 states:

> *[I]f I localise a product, the ideal case is that you do not notice that it has been localised. That's the ideal case, right? […] It does not come across as translated. (LM4)*

LM1 elaborated that some of his customers obtain localisations not out of direct concern for their international customers, but instead aim to satisfy legal requirements of providing translated content for each locale. He experienced the customer's attitude towards linguistic quality accordingly:

> *It can be said that certain companies place more emphasis on having high quality translations made. And other companies […] say, "Well, […] will not be read anyway"; there is that opinion, and [they] put less emphasis on it. That is very, very variable. (LM1)*

A similar lack of customer engagement with quality aspects of localisation were noticed by L1. As LM1 further puts it, not all customers obtain localisations for their products out of concern for their customers. Instead, they need to oblige legal requirements of providing translated content for each locale.

Notably, software quality criteria are debatable. For example, Glass (2002) lists reliability, human engineering, efficiency, testability, portability, understandability and modifiability as ultimate software quality criteria. Localisation quality could contribute only minimally, and maybe developers do not assign much attention to it for that reason.

The effect different quality criteria can have on localisation quality might be best illustrated by the worry several localisers and localisation managers had regarding the linguistic quality of the source texts they were translating. It was noted that the link between quality of source and translation is not apparent to their developer colleagues. A

translation cannot be better than the source, and if the source is unclear, translation will not be able to fix this for the target language.

4.2.1.2 Discipline-Specific Success Criteria

Interdisciplinary collaboration is also shaped by each discipline's specific success criteria, here cost, time, context-sensitivity, consistency, and localisation scope, which they bring into the overall product and process. There is a chance that other disciplines need to share these success criteria, or are involved in them in some other way. Sharing, or at least clear definition and communication of success criteria, is important as it is essential to a clear understanding of the expectations in one's work. However, Green (1994) points out that it is counterproductive for collaboration to impose one discipline's success criteria on another discipline.

4.2.1.2.1 Time and Cost

The efficiency pressure in software development has already been brought up earlier. In fact, software developers emphasise cost minimisation as a major success criterion. Accordingly, L2 experienced software projects where developers deliberately chose the cheapest translation source because otherwise they "would make less of a profit."

However, localisation seems to be even more affected by time constraints: For example, despite his company having in-country specialists for each target culture they develop software products for, D1 related how actually consulting these specialists to determine localisation requirements would take too much time for the development process:

> *Author: You do not contact the [translators] for that, either?*
>
> *D1: Well, no. No. We might have questions for [the translators], but in principle, I've, I've never had that happening. […] It takes forever. It takes way too long.*
>
> *Author: Are they so slow?*
>
> *D1: Well, there's twenty of them.*

D1 admits that there is a case to communicate with translators, in this instance to ask questions about localisation requirements. But it is not done because contacting all the translators separately takes too much time for a developer.

D3 related an experience where a complex subtitling function had to be built into a multimedia web portal because there was not enough time to translate and dub the video files directly.

LM1 related his experience of the general expectation the customers of translation agencies seem to have. In addition to translation, LM1's LSP offers proofreading services.

> *[The customers rarely book proofreading services] not because of the costs, but rather […] because they do not have time for that, or they need the translations so soon, that […] there is no time for that. That's more often the difficulty for us, because we often have to do things under time pressure. (LM1)*

Similar prioritisations of cost and time over quality were observed by developers, e.g. D10. The priority of speed and low cost in software development is widely known and its impact on quality discussed in the literature of both localisation (Papaioannou, 2005; Kahler, 2000) and software development (Boehm, 2006).

4.2.1.2.2 Context-Sensitivity and Consistency

In translation, a part of quality is determined by context and consistency. Context information, i.e. information clarifying otherwise ambiguous terms and statements, might be the single most important factor for translation quality. Its importance was discussed already in the literature review. Practically all interviews in the localiser group stressed the importance of context information for translation and at the same time had experienced difficulty in obtaining said information from the developers or customers.

Some localisers trace the lack of context information to a lack of awareness of its importance by developers. This is supported by the observation that few developers brought it up. In a rare instance, DM6 pointed out the destruction of context information by alphabetically sorted strings in Excel files. On the other hand, there is a certain implication that a business-motivated reluctance for localisers exists not to educate developers about context information.

Similarly, the importance of consistency seems to be mostly exclusive to the localiser side. LM7 elaborates:

> *Not only must this term be consistent throughout the company, it also has to work with the customer. I must be certain that when the*

> *company talks about a 'fax switch', then everybody within the company must know that a 'fax switch' is the thing that separates a telephone call from a telefax. But I need to communicate this to the customer as well. And I have to call it 'fax switch' consistently. I cannot use the term 'fax separation' in one instance, and whatnot in another. (LM7)*

Lack of context has been one of the most prominent issues in software localisation for quality (Aryana and Liem, 2011; Reineke, 2005; Forssell, 2001). Although there have been numerous technical attempts to address this lack through technology context storage in translation file formats (Bikmatov *et al.*, 2013), lack of context is still an issue. In part, this might be explained by the progress, or lack of progress, in adapting respective localisation tools. Alternatively, a lack of knowledge may be involved.

4.2.1.2.3 Localisation Scope

L1 related that in his experience, localisation scope in software projects was always constrained to language translation and related items, specifically adapting formats and units of measurement. The account suggests that in the minds of most customers, localisation is practically superimposable to translation. If pointed out, customers will understand that the scope is actually larger and expands text translation to include text-related aspects such as formats and units. However, in practice as experienced by L1, localisation scope never includes completely language-unrelated items such as colours or symbols.

This is supported by LM1 and LM4, who also never encountered instances where symbols, colours or anything decidedly beyond language had to be localised. Both expressed the belief that localisation of, for example, colours and symbols are part of comprehensive localisation, but they never encountered the need in practice. LM4 further believed that compared to his domestic customers, international customers might be more interested in comprehensive localisation beyond text. However, he explicitly pointed out that this is an assumption on his part.

On the other hand, a tendency was observed among developers to classify the localisation scope as translation only, including necessary adaptations to layout, e.g. to account for text expansion during translation, or additional voice recording for e-Learning or implementation of subtitle functionality for multimedia applications. D1 faced issues of making a software project time-zone aware, specifically having to address the fact that

some countries spread over more than one time zone. One could argue that time zones are not obviously cultural and instead can be understood as a technical property.

4.2.1.3 Tools

Interdisciplinary collaboration is shaped by the use, or non-use, of each discipline's special tools which they bring into the localisation process. Adoption of specialised tools and standards was not a topic in the interviews and were only touched upon in the context of problem causes, e.g. use of Excel or insufficiencies of existing tools. In software development, tools have been recognised as source of improvement as well as constraints (Grinter, 1995) and an influence with consequences for interdisciplinary conflicts and strategies, previously discussed in the context of increased efficiency and lowered effort (Schubert, 2009; DePalma, 2006; Law, 2003), although for some tools, a reduction in localisation quality has been suggested (Bowker, 2005).

4.2.1.3.1 Tool-Originating Restrictions

A number of tools and utilities are used in software development with the aim of making software engineering simpler, faster, more efficient, and more effective, and so on. Similarly, localisation tools with the same aim exist. However, both classes of tools come with requirements and ramifications and can have an impact on the work and activities of both developers and localisers. DM6 noted instances of this in various development tools, for example Microsoft's programming framework .Net, a collection of APIs to facilitate software development:

> *That happens comparatively often for us, yes? The dependency, we have a prime candidate here, that is .Net, because, because when using .Net we require an incredible amount of information from the code. That means this classical concept, separation of resources from code and caring only about the resources [n.b.: internationalisation], is reduced to absurdity. (DM6)*

Similarly, DM6 criticised some aspects of Java programming language API, i.e. that java.text.ChoiceFormat package allows developers to increase the complexity of placeholders by turning strings into state machines, i.e. interpreted code where the behaviour is determined by a number of states it can be in, as defined by the programmer:

> *A part of it is […] realised as state machine, the rest is free text. And all the way down at the bottom [of its documentation] it says do not use because […] translators cannot handle it. (DM6)*

On the other hand, LM6 criticised the opposite development in the Windows Presentation Foundation (WPF) from Microsoft, where developers are tempted to over-separate content into code, layout, and localisable content: an Extensible Application Markup Language (XAML) layout file for designers containing references to a XAML string table for localisers, all held together by a code written by developers. Such over-separation leads to even less context available for translation and repeated referencing of one and the same string in many occurrences for which more than one translation might be necessary.

Similarly, the usage of localisation tools can affect the work of developers. For example, many translation systems now offer an integrated visual translation environment, implementing a what-you-see-is-what-you-get philosophy where the translator can see a string within the UI while it is being translated. Often, for it to work, this technology comes with specific requirements in the way the application is programmed, i.e. following a generic standard. If a project strays from these requirements, for example by using an unsupported third-party UI framework, the visual technology can fail.

A similar issue was noticed by LM4, who related an occurrence where developers apparently expected localisers to adopt technical work and build a version control for resource IDs in addition to providing translations. Resource IDs are internal identifiers used within programs for reference and distinction of different strings. LM4 was particularly baffled because controlling resource IDs seemed to be exactly the opposite of separation from code and content, which is one of the basic principles of localisation and translation work:

> *They asked us to […] access [resource] IDs and to know what ID referred to what content. From release to release. Whereas actually we can only […] work based on text, meaning that we intentionally mask all IDs, all code, meaning that translators have only text. […] Those end up in our database, a translation memory. So, of course we have all old resources saved and archived. (LM4)*

While arguably translators working with resource IDs might be wrong to begin with, LM4's work process was prescribed by the string database tool, which would not allow certain operations, i.e. storing of string IDs.

The requirements accompanying some tools are often underestimated. LM6 noted that some of his customers, i.e. software development teams, set up their own translation memory system, but fail to maintain it. They also do not place it at the disposal of their translators, who subsequently cannot fall back on it when providing translations.

A number of accounts discussed the rectification of manual work processes, either through straightforward automation, or through a change in processes or systems that made manual steps obsolete. Interestingly, the resolutions usually address concerns of developers, e.g. inefficiencies they would have to deal with. For example, D3 describes how automation was introduced in order to avoid having to touch program code for changes to localisation. Usually what happens is that developers implement what is called a content pipeline, i.e. a series of automated work steps that brings received content in proper format to its correct location. Few accounts relate the rectification of unnecessary manual workload for localisers, including efficiency and quality issues. Many localisation processes and tools tend to evolve over time to address developer concerns because they are the ones to modify them according to their understanding and their agenda.

As was shown in the literature review, there is an almost a fanatical obsession with improving localisation and translation through the use of technology. While the successes of tools, and specifically of CAT tools, are many, there are critical views. Stoeller (2011) acknowledges that use of technology in localisation, specifically technology simplifying collaboration in translation, has enabled smaller LSPs to compete with their larger brethren. On the other hand, Stoeller also warns that technology is seen as a replacement for proper communication and other human-related issues.

4.2.1.3.2 Usage of Inappropriate Tool

Some interviewees reported usage of tools that might be considered inappropriate or insufficient for the task at hand. This was often in the context of obtaining culture-specific information. For example, D3 reported his usage of Google's MT service Google Translate

for fixing translations, and D5 reported basing locale-dependent UI design decisions on information found in an online encyclopaedia.

A recurring theme was the use of generic tools and file formats to handle translations and Unicode texts, particularly spreadsheet software to handle translations (e.g. LM7, DM6, D9). And apparently many software companies are unaware that such file formats are unsuitable for localisation (LM4). In short, spreadsheets are difficult to maintain and their use encourages inefficient and error prone manual labour such as copy-and-paste. They also discourage provision of localisation context and even lead to its removal when sorted; as DM6 puts it, "to really remove any context relation". Spreadsheets further complicate string management, too:

> *And for us it can happen by all means, those 10 Text-IDs that are completely different for developers because they appear in different places in the code, in a completely different hierarchy, for example "Auflösung", "resolution". […] [T]he Excel-sheets from predecessor projects were continued, taken over, meaning that there was just one Excel-sheet, that was stupendously huge, and the developers did not go to the trouble of looking, somehow, "I need a new Text ID now, that has text behind it, does the text maybe somehow exist already and I can simply copy the line and link in a new Text ID", instead if in doubt they just created their new text ID and wrote in "Auflösung" [German for "resolution"] for the one-hundred-and-tenth time. Right, because text ID centric work […] is easier for him, rather than somehow looking if the text already exists. With the result that obviously the Excel sheet was sent to us for the tenth time with the request to have "Auflösung" translated into all languages. […] This is to say that we really had to do it, that we simply sent these Excel-sheets to the translator and said, yes, so we simply have to translate it for the tenth time. And when I translate a text for the tenth time, there is a great danger that it is translated diversely, or differently. (LM7)*

LM7 elaborates on the difficulty encountered when using Excel files and continues:

> *What you have to keep in mind, no matter what solution you are using, regardless whether a database or Excel or whatever [is the] appreciation […] source text [vs.] translated text, […] if I change a source text, of course inevitably I will have to change the translated text. For the developer, it does not change. The ID stays the same, but the text behind it is suddenly no longer "resolution", but is called, whatever, "contrast". And the developer changes it in German, and at best in English, because he sees, "resolution" is not "contrast", but in the remaining languages have already been translated as "resolution" and remain unchanged. The tester for German-English will not see it. And if he tests Portuguese*

> *or Finnish, he will not see it either, [to know whether it should be] called "contrast" or "resolution" in Finnish, you have to be a specialist. [One must keep track of source] text changes, so that one marks it to have changed. (LM7)*

Basically, LM7 describes that the choice of spreadsheets as exchange format comes with a consequence: either a strict procedure to update source text changes and translations accordingly, or a stringent localisation testing regime. LM7 goes on:

> *Sometimes, developers […] have taken the derived Excel sheet, derived a text file from it, then made text changes in those text files, sent the text file to us, asking us to reintegrate the changes into the Excel file and translate them. It is an Excel sheet of 15 MB. The developer has it, then he changes something, he copies that somewhere, and now you have to run from one developer to another to find out who had it last, where the latest current version is, well. That's unsatisfactory for everybody. (LM7)*

4.2.1.3.3 Lack of Tools

While many different tools exist to facilitate software localisation and the development of international software, these are often not used. The need for tool support, e.g. when handling XML files or Unicode, is not always immediately obvious and has sometimes been noticed only when a project was well underway, as reported by DM4 and D10.

Lack of tools is often compensated through manual work for comparatively mundane activities such as copy-and-pasting text from one file to another. Several developers, e.g. D6 and D9, reported manual copying and pasting of translated strings into an image file in a graphics editor because the UI had been designed to display image files, not text. In some cases, this is done so that scripts other than Latin characters can be displayed without having to adapt code towards Unicode-compliance.

It does not require a lot of imagination to understand how the overhead effort through copy-and-paste can be immense when, as in the case of D6, one has to create and maintain hundreds of image files for each of the 14 different locales. Extended copy-and-paste activities like this are also prone to errors if concentration fails and a string is not highlighted correctly before copying, or pasted into the wrong file.

Further, as expressed by LM5, many "atrocious translations" are caused by the use of deprecated and unsuitable tools in the localisation process while failing to utilise state-of-the-art localisation tools specifically tailored for this purpose.

4.2.1.4 Processes

Interdisciplinary collaboration is shaped by each discipline's activities and processes, which are brought into the localisation process. Localisation is strongly affected by preceding production activities (Sikes, 2011). An example how the influence of one discipline's process might be perceived by the other discipline is given by LM4:

> *I believe the customer sometimes complicates his own life unnecessarily. Maybe due to ignorance, and even if you educate them, their internal processes are so sluggish and cumbersome that [it is difficult] to push through new things when know-how is not internal and not utilised externally. (LM4)*

4.2.1.4.1 Terminology Management

Terminology management is a significant aspect of translation, especially when translating technical documents in a corporate environment. LM7 dives into its intricacies while relating plans for creating a text and translation database:

> *[…] Structuring and separating information into modules had just become second nature for us. And it was a style of work which absolutely resonated with me personally. I think this is how one should work in a technical editorship, because it is just not about unconstrained poetic prose, but it is about structured information, which on top of it has to be well phrased and to the point. […] It was also obvious to us that the topic of [UI interface texts] was closely related to terminology management. If you think about it, it is nothing more than breaking an operation down to a single term. Not only must this term be consistent throughout the company, it also has to work with the customer. I must be certain that when the company talks about a fax switch, then everybody within the company must know that a fax switch is the thing that separates a call from a telefax. But I also must communicate this to the customer. And I have to call it fax switch consistently. I cannot use the term fax separation in one instance, and whatnot in another. (LM7)*

According to LM1, translations for technical terms or otherwise extraordinary or special vocabulary should be decided by close collaboration with the customer, but few customers are engaging in this.

4.2.1.4.2 Accommodating Changing Requirements

The need for software development projects to adapt to changing requirements is noticed during localisation. Both D3 and D11 noted detrimental effects and the challenges of repeated changes to application requirements and localisation scope, especially when

localisation requirements come into a project when the overall software architecture already is in place. The account of LM3 seems to suggest that this is a consequence of a trial-by-error approach to software development:

> *So, you know, it is basically, a lot of companies are like that. They will not really listen until they have an incident where it really affects them. And then they understand what the issues are. (LM3)*

Similarly, L1 suggested that the trial-and-error approach permeated in software development: software testing is an integral part of software development, which is reflected in the development process. Developers follow a trial-and-error paradigm as "optimistic technologists" (Green, 1994, p.328) that cannot work for localisation as linguistic testing in most projects does not take place.

4.2.2 Conflicts

When working on international software, all interviewees sooner or later experience a number of conflicts. These are rarely personal conflicts. Rather, these are conflicts created by actors in different organisations following different guidelines and using different tools in order to achieve different goals, although they might share an overall end. Figure 4-5 gives an overview of the specifics concepts in this category.

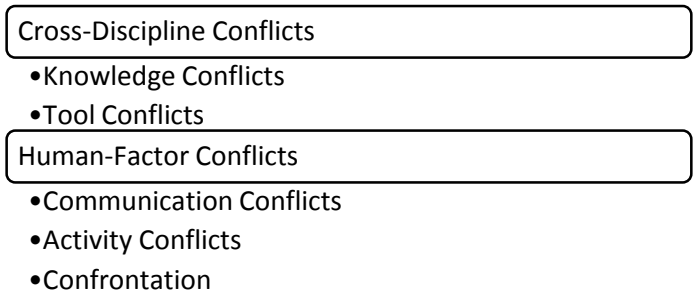

Figure 4-5 Emergence of the category Conflicts

Such social conflicts have been characterised by Thomas (1992) as a dynamic development that can either be constructive, e.g. by initiating changes and clarifying relationships, or destructive, by escalation and standstill. Unfortunately, existing empirical studies suggest that overall, conflicts go down the destructive path (De Dreu and Weingart, 2003).

4.2.2.1 Cross-Discipline Conflicts

Both localisers and developers encounter a number of social conflicts in their work through their knowledge and tool use when these are influenced by their discipline.

Although social conflicts often include some kind of direct interaction, they often appear to be hidden and participants seem to be unaware of either the conflict itself, or its source, or its ramifications. LM4 reflected about it in the following way with respect to an unsuccessful localisation project:

> *I believe [the project] failed due to certain individuals, who maybe sometimes were beyond their abilities, or who asked for things we could not deliver, where you often talked past each other, and eventually everyone involved was dissatisfied. (LM4)*

4.2.2.1.1 Knowledge Conflicts

Some conflicts appear to have their roots in the often unspoken assumption that one's own knowledge is shared by all collaborators. But in some cases, it is the outright and known lack of knowledge awareness cannot remedy that causes problems. For example, a few developers assume that translators can understand certain aspects of source text formats, e.g. that they can interpret XML files or recognise and leave alone C-style placeholders. However, these technical skills should not be assumed.

From the point of view of localisers, the most apparent knowledge gap is that developers do not know what precisely localisers, translators and technical writers are doing in the first place, and what happens during the localisation process:

> *The problem is that developers often do not even know what I am doing, or what information I need. I sometimes suspect that I know more of the technical process than they do, and that is bad, because I have no technical background. (LM2)*

DM3 attempts to describe the knowledge difference from a more neutral point of view:

> *The persons who do the translation, they do not sit well on both sides of the barricade. They either... they may be fantastic writers, they may be linguists, extraordinary linguists, but then they do not understand usually the nature of the application or the system they try to localise. (DM3)*

This lack of knowledge regarding localisation as activity and process is probably most prominent in the ignorance about consequences a lack of context has on translation quality, which is also discussed as a discipline-specific quality criterion. LM2 illustrates this relationship between lack of context information and lack of localisation process knowledge when relating what the biggest problem during localisation is:

> *No doubt number one: No context. It has improved with those [developers] I spoke directly to, who were up here and saw how I translate and how it works. Because in meetings, it is often said that I, "he up there in software localisation", am just not good at it. And those [developers] who learn how I work then say, "he can't, because he does not see it [in context]." (LM2)*

Similar observations, i.e. developers lacking knowledge and awareness about linguistic processes and requirements, were made regarding a wide range of localisation issues, such as a lack of awareness regarding the link between source text quality and translation quality, the need to manage terminology, or how localisation prices are calculated.

By the end of the literature review, among others, two problems had been identified: a lack of cultural knowledge by developers, and missing integration of localisation into the software development process. Participants of the localisation process have expressed a preference for cultural knowledge for developers in this research as well as other empirical research (Immonen and Sajaniemi, 2003a, 2003b). In fact, some participants on the developer side did mention efforts to acquaint themselves with cultural knowledge or integrate cultural sensitivity into their work products, particularly if the frontend or UI was part of their responsibility. Repeatedly, the previously discussed model by Hofstede was mentioned in this context. On the other hand, some interview accounts showed that existing cultural knowledge was apparently not necessarily recalled when necessary. For example, DM7 had witnessed a native English-speaking product manager who, without consulting translators or DM7 himself, planned to substitute names in UI strings automatically without human review despite conflicts with English grammar[46].

Although the interviews gave examples where lack of cultural knowledge of developers caused localisation issues, it appears now that understanding the activity of translation

[46] Specifically, the planned substitution was going to introduce errors where the indefinite article "a" would have had to be changed to "an" because the new name started with a vowel.

plays a larger role. This has been implied before. For example, the dependence between translation quality and the source quality is underappreciated (Bauer and Rodrigo, 2004; Russo and Boor, 1993), and Combe (2011) suggests that a lack of understanding of localisation as a process is responsible for collaboration problems between localisation and development.

4.2.2.1.2 Tool Conflicts

During the work on their tasks, localisers and developers use various tools, which are already considered as an influence on the localisation process. At times, tool usage can also be conflicting.

DM6 noted how API frameworks at times conflict with internationalisation, i.e. the idea of strictly separating code and content, or other linguistic requirements such as provision of context information or consistency.

There seems to be an overall push towards using localisation tools that allow the localisation of compiled binary files directly, thus avoiding the file exchange problem completely. However, tools for localising binary files directly come with other risks. LM2, using one such tool, noticed a discrepancy between the representation of UIs in his localisation tool compared to the developer's product or the final software.

> *Even if it looks good in my tool, when the developers have it on their screen, it might be that it does not look right anymore, and then they come to me and I have to change pixels. (LM2)*

Basically, he has to guess what his adaptations will look like in the final product. Originally, the idea had been for the translator to see and translate text in the context of the UI (see e.g. Freigang, 2000; Esselink, 2006). However, what is now the problem is that the localiser could not reliably edit the context of the string, i.e. the position of the UI elements. The expectation to have localisers edit the UI indicates that the role shift for localisers from mere language processor to editor, as described by Yuste (2005), is already in progress.

On the other side, interviewees experienced difficulties with combining localisation tools and processes with software development tools. DM7 related his considerations whether string translations should be stored in a source code repository: On one hand, it is

desirable to store all sources and resources centrally so that any changes can be tracked and any version be rebuilt. On the other hand, DM7 noted that storing translations in a code repository is incompatible with storing and maintaining them in a translation memory.

In the literature, it has been noted how localisation and internationalisation APIs have simplified software localisation (e.g. Immonen and Sajaniemi, 2003a; Kalliomäki *et al.*, 1997). It is noteworthy that many issues reported in somewhat older publications, e.g. around string length limits, code page and character encoding (e.g. Law, 2003; Carey, 1998), were not mentioned during the interviews, suggesting that certain standard developments, e.g. the ongoing improvement and proliferation of localisation APIs and Unicode, has indeed improved localisation. However, considering the localisation scope in the literature and the influences of tools, and here specifically APIs, one wonders if today's localisation scopes and localisation requirements are determined by the API. On one hand, when referring to the localisation requirement survey suggested by Kalliomäki *et al.* (1997), there can be fewer illustrating examples how much APIs have simplified localisation in the last 20 years as many aspects of the survey, e.g. regarding encoding, internal data representation, and locale-independent operating system functions, have become moot. On the other hand, no participants reported elaborate localisation requirement elicitation processes as described in the literature, e.g. a translatability and market suitability analysis described by O'Sullivan (2001a), and one wonders about the impact following localisation APIs blindly can have for product quality.

4.2.2.2 Human-Factor Conflicts

Whereas cross-discipline conflicts were directly related to the collaboration of two disciplines, some conflicts appeared to be more related to general human aspects of collaboration, even though these appeared to be shaped indirectly by the different disciplines and their relationship between or within organisations.

4.2.2.2.1 Communication Conflicts

Communication between translators and developers is not always smooth. Some localisers feel that developers do not like to communicate much to begin with. Communication further appears to be prone to error, especially when it is indirect. L1

explicitly contrasts communication through an LSP with direct customer communication: In the first case, localisation issues forwarded through translation agencies have a “fifty-fifty chance” (L1) of the LSP bringing it up to the customer. If they do not, there is a curt reply to translate what’s written and not bother about the rest. Anyhow, L1 describes communication through intermediaries as “incredibly long and complicated” (L1) and leading to loss of information along the way.

The failure of communication via LSP is also noticed by developers. The following misunderstanding was reported by D2 when writing software for the control of 15-segment LED displays. D2 was acutely aware of the limitations of such displays, in particular when it came to translations:

> *D2: With 15 segments you can get letters as well. You cannot get any accents, […] there's nowhere there for a sort of an accent on a letter. There's just no possibility at all, […] there's no chance. And the problem there, one, in the Spanish, was a bit when we got a bit about the date. Because the Spanish translator sent things back with tilde on, and so on. And I had to say, no, you know, it is not that we do not want to do that, it is we can’t. […] And, for example, year is “año”. [Writes año.] That's how you spell “year” in Spanish. So I was saying, no, not with that you can’t. On that. [Strikes out the diacritic so that it now says ano.]*
>
> *Author: And did the translators agree that this is still acceptable?*
>
> *D2: Well, no. I was going through the agent, and the agent said, [he has] to check with the translator. And the translator came back and said, that's totally unacceptable because that spells differently and has a different meaning.*
>
> *Author: Does it?*
>
> *D2: Well it means... [Points into his rear.]*
>
> *Author: Does it.*
>
> *D2: Yeah. [laughs] This is why it was totally unusable. […] There was a communication problem first. Because. You know, I did clearly say, you know, we cannot do accents. […] And then I get a response that is with accents! There was a fault in communication there. […] But all that... obviously I had not got that message all the way down to the person that needed to hear it. […] I thought I explained it properly to the agent. But, between me explaining it to the agent and the agent explaining it to the translator, you know, something got lost. And having two stages, I never spoke to the translator.*

Vis-à-vis, LM2 reported:

> *LM2: We once had Russian, that was completely Unicode, no context information whatsoever. On top of that with length limit. And as length limit, the colleague told me, it could be line-wrapped*[47]*. And that's what I communicated on. And [the translators] really did it one below the other, the same way I understood it. But what the colleagues had had in mind was "blablabla-hard hyphen" in one line, and after the hard hyphen some more text. A complete misunderstanding.*
>
> *Author: Just that I understand this right: In the mind of the translators, it was a hyphenated break?*
>
> *LM2: Exactly. But it was in one line. Now, if somebody says "line-wrapped" to me, I'd interpret that as one below the other. With a line break. But I was not told. So I did not communicate it on correctly. And then the problem duplicated into all languages.*

Communication does not always have to break down. At times, it is simply a matter of what needs to be communicated. LM7 describes this while relating an instance where he needs to talk to developers about a specific piece of text in software:

> *LM7: [T]he developer knows his [resource] IDs. [You ask] him: "Listen, the ID yadda yadda, XYZ, what did you mean by that?" And then he would know […] that is in the code at that location, and then he knew [the answer]. For them, communication via IDs is important. You have to mention them. […] And then he usually already knows. Or he quickly looks in the code and finds it.*
>
> *Author: So, […] that implies […], if you had asked the developer [about] a button called 'Select', then he would not know what you are talking about? […]*
>
> *LM7: That happened, yes. [re-enacts conversation]*
> *"You have a text in the web interface, that is called […] 'search LAN network', or something like that?"*
> *"Yes, where is that?"*
> *"That's in the third mask in the lower left."*
> *"Well, no idea."*
> *"It has the resource ID so-and-so."*
> *"Oh, right, wait, I'll check quickly."*

D3 notes a similar experience, but from the developer side: text changes are communicated by giving the original and the changes to developers, but the developers

[47] LM2 uses the word "zweizeilig", i.e. "two-lined" (Scholze-Stubenrecht et al., 2005, p. 850). Here, the translation "line-wrapped" was deemed more appropriate due to the focus on the hyphen.

have to identify the text's location within the application or source themselves, usually by searching through an Excel file.

LM5 summarises the communication issues as follows:

> *It is a fundamental principle that the two disciplines cannot communicate with each other if they do not acquire a common register. […] But it is common that the disciplines do not understand each other, meaning that translators do not understand anything from the developer and the other way round, and the project manager does not understand why the developer reacts in a certain way, and the other way round, and so on. So there is only one way to resolve this, by developing a common register. (LM5)*

These excerpts illustrate how fraught with misunderstandings and communication obstacles the communication between developers and localisers can be, and further how these lead to localisation issues such as incorrect translations and delays.

The so-called *standard view of communication* understands the communication process as exchange of a message encoded by the sender and decoded by the receiver. This primarily technical model explains communication difficulties due to lack of precision of the message and message degradation during transmission (Thompson, 2003). Such degradation could occur for example though indirect communication via intermediate managers as described by some interviewees. Other than that, if the message is precise and not degraded during transmission, the model does not account for the failure of communication observed here.

Barnlund (1970) proposed a *transactional model*, which accounts for reciprocal communication between two sender-receivers. Barnlund's model considers that communication includes the encoding and decoding of so-called *cues*, some of which are public yet can be modified by the environment, some of which are private and depend on each sender-receiver's understanding of the world, their experiences, education and so on. If the sender-receivers' schemata for encoding and decoding are not identical, then the decoded message does not hold exactly the originally encoded meaning and communication is compromised. Some of the accounts on communication from the interviews seem to exhibit this issue, and a common understanding of the cues might be what LM5 refers to as "register". Green (1994) examining the interdisciplinary collaboration of psychologists and software engineers, suggested to form an interface

between psychologists and developers by creating a boundary vocabulary specific to this collaboration.

In software development, communication has been identified as an important and crucial aspect (Brooks, 1995). Perry *et al.* (1996) examined more formal aspects of communication of software developers and found that they spend up to 75 minutes a day communicating, mostly in frequent, quick, informal and improvised exchanges. The study also found developers to be shunning email because it is perceived as a broadcast medium, does not fit with an iterative resolution process required by software development, and simply is slower than just walking over and talking. Informal communication seems to be subject-specific to software development. Herbsleb et al. (2000) found that specific software development aspects, e.g. requirements documentation and requirements changes, are commonly communicated informally because of the unsuitability of formal means. Similarly, Herbsleb et al. (2001) found that in software development, distance decreases communication and increases delays. Specifically, communication in cross-site projects takes longer compared to same-site work, and requires more people. Additionally, in cross-site projects participants are less likely to receive help overall and more likely to experience delays. Partly, this might also be explained with the lack of informal communication in cross-site projects.

Such a finding implies that software developers communicate more if they can simply walk over and talk, i.e. that communication distance and communication volume correlate. In fact, Herbsleb *et al.* (2001, 2000) and Espinosa *et al.* (2002) showed the positive influences of co-location on team coordination and development time. Allen (1977) reports that with increased distance between engineers, communication frequency decreases, up to a distance of 30 metres between offices. At increased distances, even up to miles, communication does not significantly drop any more. Herbsleb *et al.* (2001) compared same-site and cross-site projects, i.e. projects where developers were co-located and projects where developers were spread out over several sites. The study found that despite constant willingness to help in both kinds of projects, developers in cross-site projects are less likely to receive help and are more likely to experience delays getting help. The authors concluded that help over distance is relatively ineffective. Accordingly, Herbsleb *et al.* (2000) presents a number of examples from

various disciplines showing how co-location improves collaboration, efficiency and quality of work even when this was not the primary reason to co-locate workers.

The particular importance of communication between different roles in software development has been emphasized by Brooks (1995). Accordingly, the localisation literature abounds with the need of and recommendations for constant and direct communication between developers and localisers (e.g. Giammarresi, 2011; Bauer and Rodrigo, 2004; O'Sullivan, 1989).

4.2.2.2.2 Activity Conflicts

Sometimes, best practices of one discipline are in conflict with another: In programming, resource items can be any number of things, often parts of the user interface such as an image, a dialog, or a link. Each resource item has a unique ID, which is used in the program code to refer to it. As such, the actual item behind a resource ID is not directly accessible or visible in the code. This is good practice in software development because the actual resource items do not have to be finished at the time the code is written. It is also good practice with regards to localisation as items behind a resource ID can be replaced at run time in order to display values for different locales as required.

Often, cross-discipline activity conflicts seem to be related to discipline-specific processes. For example, LM4 noticed final content sent for translation, then being changed and sent for translation again, incurring unnecessary cost. This might well be a consequence of iterative development, where it is difficult to find a place for translation (Combe, 2011). DM7 noted that because his projects' development process did not freeze strings that had been sent to translation, it happened regularly that belatedly changed content remained untranslated in the published version, causing serious customer acceptance issues in areas where Latin script or English language stands out, e.g. in South-East Asia. Similarly L1 noted that the trial-and-error approach permeates software development: software testing is an integral part of software development, which is reflected in the development process.

The conflict between translation and its need for static documents, and agile development processes and its aims to eliminate static documentation or products, has been discusses by Combe (2011), who noted the impact of agile processes specifically

with regards to translation accuracy and completeness. To facilitate translation nonetheless, large content volumes have to be processed late in the development cycle instead, increasing schedule pressure on localisers.

However, activity processes are more general, as related by LM4, who noted an occurrence where developers apparently expected localisers to adopt technical work and build a version control for resource IDs in addition to providing translations. This particularly baffled LM4 because it seemed to be exactly the opposite of separation from code and content, which is one of the basic principles of localisation and translation work.

> *They asked us to […] access [resource] IDs and to know what ID referred to what content. From release to release. Whereas actually we can only […] work based on text, meaning that we intentionally mask all IDs, all code, meaning that translators have only text. […] Those end up in our database, a translation memory. So, of course we have all old resources saved and archived. (LM4)*

The project did not end well:

> *I believe it failed due to certain individuals, who maybe sometimes were beyond their abilities, or who asked for things we could not deliver, where you often talked past each other, and eventually everyone involved was dissatisfied. (LM4)*

Similar to communication conflicts, activity conflicts quickly lead to delays, and further to quality issues such as bad or missing translations.

In fairness, it is misleading to suggest that developers are not interested in improving localisation quality, e.g. by answering questions or providing additional information. In some interview accounts of localisers, developers are very forthcoming, answer questions, provide additional information, or even proactively inquire about the cultural compatibility of their software. In others, they are not helpful and will not even look up existing terms before submitting new ones for translations. In yet others, they simply seem to be clueless. DM7 told of an incident where he had pointed out to a colleague that placeholders in strings require additional context information so that translators know how to translate them. Instead of writing any documentation on the placeholders, the developer in question instead split up the strings a the placeholders and concatenated them at runtime with variables, believing that he had solved the problem, when he had made it worse since the context of the partial strings was now also missing.

A major aspect seems to be whether developers and localisers work for the same organisational entity. Unsurprisingly, developers and localisers working for the same company seem to communicate more, possibly because they share the same business goals.

This becomes most apparent when developers and localisers do not work for the same business unit, i.e. when localisation is not conducted in-house, and they may have differing business goals. A software company, after all, aims to deliver software. Localisers might be more interested in satisfying their own customer expectations, as opposed to the expectations of their customers' customers. LM4 sums up the expectations of localisers' customers, i.e. developers, as follows:

> *My impression is that […] [developers] like to have it the following way: "Here you have the stuff, work with it, and we want finished language packs back, and in the best case we have nothing else to do with it." (LM4)*

LM4 even tried to improve the localisation process by pointing out improvement opportunities:

> *Well, when I addressed [causes of recurring localisation problems] with some customers [i.e. developers], I got excuses or stories that they cannot do it any other way. Naturally, in the end, we do what the customer wants. Although I do not quite understand why they want it. (LM4)*

In the end, LM4 concentrates on his bottom line by satisfying his customer's expectations.

Another example of localisers concentrating on their business interests might be the proliferation, or lack thereof, of knowledge about the translation process. While previously it was established that software developers do not know about the importance of context information for successful translation, in an outsourcing environment, there seems to be reduced incentive to actually educate them about this problem, as noted by LM5:

> *Author: [D]o translators bring up […] that they need to know the context?*
>
> *LM5: Less. They look after lines of text and words. That's what their rate goes by.*

L1 supports this by pointing out that inquiring about context and conducting research eats into his net rate. Since freelance translators and LSPs are usually for words instead of time, there is little reward for them to educate their customers towards a better quality. L1 also reported finding context-related errors in existing translations where other translators had obviously taken a guess when translating. When L1 inquired whether those translators had ever pointed out the problem with the source text, the LSP claimed that it had never been informed by other translators about the issues.

It appears that there can be a conflict of interest for localisers, both freelancers and LSPs, due to their business goals, manifesting itself here as lack of information exchange. Dunne (2011) refers to this as "temptation - or pressure, as the case may be - to allow urgency to trump other constraints when developing and managing project schedules and to move into the realm of post-heroic project management" (Dunne, 2011, p.149).

Although this is not a communication problem per se, applicable theories exist in the area of communication in work organisations. According to Tubbs and Moss (2003), Dennis (1975) describe four categories of organisational communication: Downward, upward, horizontal and informal communication. Horizontal communication refers to communication between departments for the purpose of task coordination, problem solving, information sharing and conflict resolution, and should be the communication mode between developers and localisers. Although horizontal communication itself can be affected by rivalries between inter-organisational entities, it may be dramatically affected if hierarchy is perceived and it changes to upward and downward communication. If for example the localisers should perceive developers as hierarchically higher, then their upward communication is informed by what is in their own interest. This would explain the observed communication differences between in-house and outsourced localisation scenarios.

4.2.2.2.3 Confrontation

During localisation, developers and localisers can experience confrontation. Grinter (1996b) suggested that for each technical dependency, a social relationship exists that needs to be managed. This social relationship at times can become personal. LM2 relates that when localisation issues are discussed "in meetings, it is often said that I […] am just not good at [localisation]". DM7 felt that developers are even more likely to resort to

blaming when localisation is outsourced. Such attitudes seem to have consequences. LM7 suspected that asking for missing context information might be interpreted as being clueless.

LM3 felt that developers appear to worry about control over software. In discussions on cultural awareness he has to convince developers that he's not the political correctness "police" and not one of the "suits" (all LM3) either, but a confederate of developers with a passion for software development. A case study reported by Tuffley (2003) resonates here, where a technical writer had to create the requirements specifications for a developer of a non-global software development project. The developer perceived the technical writer as a threatening influence and was initially unwilling to collaborate.

Some publications suggest that it can be difficult to establish a working relationship to software engineers (Cooper, 2004, p.106), who at times might be unwilling and "recalcitrant" (Combe, 2011, p.321). Deal and Kennedy (2000) have identified four predominant types of organisational cultures, referred to as tough-guy-macho culture in risky and dangerous settings, work-hard-play-hard culture in financially competitive companies, bet-your-company culture in businesses with large upfront cost and slow return, and process culture in bureaucracies or companies with little risk and little return.

The process culture as type of organisational culture model might apply most closely with the situation of a freelance translator. Translators are relatively remote from the results of their work and the LSP acts like a bureaucracy. Developers, on the other hand, might work in various different organisational cultures. Tubbs and Moss (2003) list IBM as example for an organisational work-hard-play-hard culture, and NASA and Boeing as examples of organisational bet-your-company cultures. Software development plays a major role in each of those examples.

Hence, unless a localiser's client happens to be situated in an organisational process culture, the developer-localiser collaboration is likely to take place between organisational cultures. On top of that, as different developers might work within various organisational cultures, the nature of the cross-organisational-culture collaboration is prone to change with each client. From a localiser's point of view, not only is the client's organisational culture different, but chance is that the nature of the difference changes with each client.

4.2.3 Strategies

It was established earlier that at the core of interviewees' experiences and efforts is the facilitation of interdisciplinary collaboration, either through defining clear interfaces, processes and deliverables, or by establishing direct interaction. Accordingly, the strategies employed and reported by interviewees could be distinguished between integrating localisation into the software development process, and separating the two. Figure 4-6 gives an overview of the specific concepts in this category.

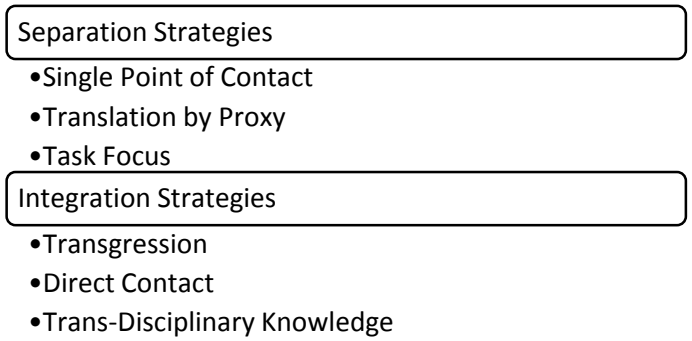

Figure 4-6 Emergence of the category Strategies

In the context of configuration management, although software engineers tend to view them as fundamentally technical, "software dependencies are […] relationships among code, people, and organizations that have technical and social aspects" (Grinter, 1996a, p.3). In other words, behind each technical dependency, there is a social dependency. This also holds for localisation. Internationalisation is a way to manage technical dependencies, but leaves the social dependencies unaffected. There is still the aspect of collaboration, described in this theory. This view supports the notion that the actual issues in software localisation are actually "existing gaps in the organizational structure and competencies" (Giammarresi, 2011, p.22) or manifestations of the so-called "silo effect" (Sikes, 2011, p.262).

4.2.3.1 Separation Strategies

Many activities and strategies employed by developers or localisers cause or imply working separately within each discipline. Several interview accounts reveal a lack of collaboration and communication, particularly for linguistic work such as provision of style guides or terminology management, supporting claims in the literature, e.g. DePalma (2006).

Arguably, separation starts with the practice of internationalisation. Hudson (1997) derives the separation of localisation and software engineering processes from internationalisation, what originally should have been an architectural choice to allow separate maintenance of code and localisable content. O'Sullivan (2001b, p.5) distinguishes between process model and architecture model of software localisation and recommends the latter because it would presumably be more difficult to adapt work activities and processes. So, separation strategies are motivated by a desire to keep working as if other disciplines were not included. Software architecture becomes a stand-in for process. This strict separation and no or minimal communication has been called the mono-directional mode of technical localisation (Schubert, 2009) and tends to reduce localisation to translation (He *et al.*, 2002).

However, localisers also try to separate their concerns. LM7 at first considered to integrate the UI designers and software developers into their workflows and give them access to the database, but soon decided against it:

> *Originally, it had been considered that software developers can browse the database whether the text they are looking for already exists. But we quickly said goodbye to that idea and thought, "No, we rather do it ourselves, it will not work." [laughs] […] [They] had not gone to the trouble of looking for existing strings within the Excel-sheet [n.b.: the previous way of storing strings and their translations], so looking through a database or so, they simply did not bother. (LM7)*

4.2.3.1.1 Single Point of Contact

Almost always, outsourced localisation has one or two sequential single points of contact handle all traffic between developers and localisers, including communication and documents. LM4 describes the translation process from the point of view of an LSP:

> *In the function as project manager [...] I maintain the relation to specific customers, and organise all the projects from the beginning. We get the files from the customer, we make a quote, we select the respective translators and we hand out the files and work to our freelancers and also small agencies. (LM4)*

This relay communication is retained for any communication:

> *Our point of contact, we [...] get questions from our translators, I collect them, send them to our contact person on the customer side, and that person traverses the company and asks the developers or so, and the*

> *customers answer from the side of the developers. [...] Especially with regards to technical questions, and that is important, and that works quite well. Most of the time. (LM4)*

LM4 anticipates one of the detrimental consequences of such a single-point-of-entry strategy. It can easily lead to disadvantages, specifically regarding the issue of translation context and how to obtain it. Nonetheless, LM4 sees the approach working well. This view is supported by LM1, working at a different LSP. He feels that there is little to no need for translators and customers to communicate:

> *As a rule, everything is clear, those are all translators that know their area of expertise very well, bring technical competence, and very rarely have any questions. (LM1)*

On further inquiry, LM1 subsequently implied that translation errors related to lack of context are generally caused by faulty source texts, particularly texts not written by trained technical writers.

Similar setups were reported by other interviewees. L1 describes such communication through intermediaries as “incredibly long and complicated” and leading to loss of information along the way. D4 had worked in projects were all localisation questions were answered by project managers, who were neither translators nor localisation specialists. The exact workflow might differ from LSP to LSP and apparently even from customer to customer.

L1 sees the single point of contact as strong contrast to direct customer communication: With single points of contact, localisation issues forwarded through translation agencies have a “fifty-fifty chance” (L1) of the LSP bringing it up to the customer. Otherwise, L1 suspects that the LSP does not even approach the customer because the translator merely receives a curt reply to translate the source material and not bother about the rest.

In software development, the dangers of indirect communication is known. For example, in the related discipline of requirements engineering and eliciting respective information from customers and users, Paetsch *et al.* (2003) recommend to avoid knowledge transfer chains and talk to the responsible persons directly. Nonetheless, funnelling

communication between developers and localisers through a single point of contact is a recommended approach (e.g. Sikes, 2011; Combe, 2011).

4.2.3.1.2 Translation by Proxy

The separation of disciplines and the single point of contact is further enhanced by a strategy where a whole organisation is behind the single point of contact between developers and localisers. This setup is intentionally and explicitly implemented as bottleneck:

> *[T]hat is a bottleneck. And that's how it is supposed to be. That is, it is a bottleneck on our side, and ideally on the customer side also a bottleneck, let me say it like that. One person who is responsible, on the customer side, for communication with us, and who has all the know-how connected to translation, and virtually is in touch with the developers on customer side, and marketing, and so on, product leader, product manager, whatever. And, well ok, the context information [...] is hard to get by for software. (LM4)*

Accordingly, L1 characterises the role of translation agencies from his point of view as freelance translator as an organisation that takes the software out of their customer's hand, extracts the strings, brings them into a translatable format, patches them back into the code, and sends it back to the customer. In some instances, part of the workflow might be executed by the customer. In each case, the LSP selects a translator, decides the relevant technical aspects such as what translation memory tool and exchange format is used in the communication between translator and LSP, and finally has the translator translate the text.

There are variations to translation by proxy. For example, instead of translation agencies, some interviewees (e.g. D3, D11) reported that translations were provided by the customer organisation that had commissioned the software as well. However, there are complaints with such a setup, as it does appear to cause delays. Comparing projects with employed translators, D3 found that proxy translation "takes too long".

In a proxy setup, the work patterns of developers and localisers likely differ considerably. The former have a nine-to-five job, whereas translators work task orientation regulated from the outside (Haralambos *et al.*, 2004). Stoeller likens translations by proxy via translation agencies to an assembly-line approach to localisation "where team members

rapidly switch from project to project to complete individual tasks" (Stoeller, 2011, p.308), lacking the big picture, i.e. the product, and losing commitment to and engagement with quality and client in the process.

As the interviews confirmed, localisation outsourcing is quite common. It is usually done because localisation is neither the core concern nor the core competency of software companies and developers (Esselink, 2006; Yuste, 2004). Outsourcing often comes with single points of contact and translation by proxy wrapped in one package. Unfortunately, such constellations run the danger of focussing on deliveries and neglecting processes (Stoeller, 2011). In such a business system one wonders whether the collaboration with the client is informed by a traditional work regime in the Weberian sense: Max Weber distinguishes between traditional action, motivated by established custom, and rational action, continually re-examined towards increasing efficiency (Gerth and Mills, 1991; Haralambos *et al.*, 2013). While there certainly is a continuous efficiency pressure within a software company or an LSP, it appears that the work organisation with the client is more traditional.

4.2.3.1.3 Task-Focus

As discussed earlier, internationalisation is often viewed as either separation of locale-dependent and locale-independent software elements, or as developing a culture-neutral software core, or finally as designing software to be configurable for various locales.

Some interview accounts suggest that developers seem to turn a blind eye on issues that they feel are outside of the technical domain.

> *Most of the developers have a feeling if something goes wrong [in localisation]. […] At least for crass blunders. I think a basic sense for it exists. Certainly not in its intricacies […] But I believe that they notice when something goes badly wrong. But it is also my experience that in their mind, it is clearly not their task to work on it. (LM7)*

At the least, developers simply do not want to be engaged with localisation, as also noted by LM4. Similarly, L2 mentioned that a collaborating developer straightforwardly stated his lack of interest in localisation quality. For him, only software quality mattered. This manifested itself by a focus on the code part of internationalisation, e.g. how to load different resources into the software. Apparently, the developer felt that he was

accountable for code, not for translation quality or required effort, for which he refused to contribute.

In a variation, some developers prioritised adherence to development processes over localisation quality despite known quality problems. The conflicts between development process and localisation quality were noticed, but according to the motto 'it cannot be what must not be', acknowledged correspondingly by remarking that the process had been signed off by the client.

In all of this, it needs to be considered that developers are usually also the ones who decide on internationalisation scope and processes. If not individually, then as a group, they draw the line that they later consider the border of their responsibility.

Losing sight of client and user concerns in software development is not a new phenomenon. In fact, customer inclusion is what agile methods have been designed to combat (Winter and Rönkkö, 2010; Vinekar *et al.*, 2006; Larman and Basili, 2003).

4.2.3.2 Integration Strategies

In software localisation, the practice of developers and localisers benefits considerably from unencumbered communication and direct access. LM2 hinted a number of times at the difference between association and disassociation, when developers "were up here and saw how I translate and how it works" and who "learn how I work" as opposed to when developers "do not even know what I am doing" (all LM2). The benefits were also clear with regards to communication and scheduling. Several authors have pointed out the need for direct collaboration between developers and localisers (e.g. Anastasiou and Schäler, 2010; Law, 2003; O'Sullivan, 1989). For translators, the ability to work with professionals of other disciplines and be integrated, ideally on-site, is clearly beneficial (Albir and Alves, 2009; Yuste, 2004).

4.2.3.2.1 Transgressions

A common occurrence during software localisation is a transgression across disciplines, i.e. that a member of one discipline does work that in principle belongs to the other.

Often, developers decide on the scope of internationalisation and localisation, e.g. D5 basing internationalisation decisions on entries in online encyclopaedias. Other reports

have developers set the localisation schedule without feedback from actual localisers (LM2), and several participants worked on projects where developers had conducted linguistic testing of translated strings without defined quality criteria or an understanding of translation testing. D3 conducted translation fixes for languages he did not speak using online translation tools. Last but not least, a practice so common that it does not raise any eyebrows is the creation of texts by developers rather than linguists or technical writers (LM1).

It appears that transgressions can also be indirect, if developers apply software engineering paradigms on localisation, e.g. by equating translations with resource IDs (L1). In some processes, though, resource IDs were generated by technical writers and had to be implemented accordingly by developers (LM2). LM2 pointed out that he feels he knows more of the technical localisation process than the developers.

In a way, transgressions can be considered a contradiction to internationalisation as the concept of separating locale-dependent from locale-independent aspects of software while equally separating development and localisation activities. It is not always clear why transgressions occur. In the interviews, it often seems like a matter of convenience or opportunity and not worth the hassle of contacting another discipline about.

Immonen and Sajaniemi (2003a, p.161, 2003b, p.30) have found a preference in engineers to conduct translation tasks if they know the language. They suggested that this is in order to save cost; this is arguably unconvincing as a developer's time should be too valuable for such tasks. Instead, when developers take over tasks that translators should or could do, it might be a lack of appreciation of translator skills, or even a drive for control.

Transgressions can, however, also have serious consequences. For example, an incorrectly planned internationalisation can pervade the remainder of the entire project or require expensive re-engineering. It can also obscure the goals and criteria of successful localisation. In an account of DM5, a usability expert had been tasked with responsibility for localisation, but conducted these according to usability principles instead of localisation principles, for example by emphasizing consistency despite cultural inappropriateness.

Transgressions of localisers were also reported, often in the shape of adapting UI layouts to handle text expansion through translation, left-to-right languages or fond size increases for Asian languages. Hartley (2009) predicts that engineering-related tasks would increasingly be handled by translators, and this might be what Yuste (2005) referred to as translators slowly taking over activities of editors. However, DM7 pointed out that localisers editing UI interfaces, even in visual editors, is problematic because sometimes interface layouts are complex and require knowledge of the underlying code to edit them. UI elements may overlap and be aligned so that only one element at a time is visible at run time. In interface editors, such aspects are not apparent. Accordingly, Anastasiou (2009) assigns such tasks to localisation engineers.

4.2.3.2.2 Direct Contact

An apparently very important factor for the shape of the collaboration of developers and localisers is the possibility of direct contact. Interviewees related different degrees, ranging from tele-communication via email or phone to face-to-face meetings. Some accounts suggest that it is particularly effective when developers can observe how localisers work. The limitations of localisation tools or the need for translation context then becomes obvious, as this remark by LM2 illustrates:

> *[The lack of context] has improved with those [developers] I spoke directly to, who were up here and saw how I translate and how it works. Because in meetings, it is often said that I […] am just not good at it. And those who learn how I work then say, "He can't, because he does not see it [in context]" (LM2)*

Accordingly, interviews who were able to directly contact developers or localisers respectively noted this as a great boon to their work, and most interviewees working in the same organisation with collaborators from other disciplines usually treated direct access as an opportunity for efficient, speedy collaboration. Direct communication does not only allow localisers and developers to ask and answer questions and avoid miscommunication, it also simplifies feedback and thus process improvement (Sikes, 2011).

The effects of co-location, direct contact and open communication have been examined in the context of teamwork and team performance, where a team is a group of professionals consisting of at least two persons and working in direct interaction on a

common task over a longer period of time. Among others, teams have been attributed with accomplishing tasks beyond the reach of individuals due to increased motivation and commitment of team members. Research shows that team work becomes more efficient as a feeling of togetherness forms (Tuckman, 1965). Seeking direct communication between developers and localisers, e.g. LM2 inviting developers to see how she is working and what defines her work, might therefore be seen as an attempt to form a team, or at least to become part of a group.

Regardless of in-house or outsourced localisation, no interviewee described a collaboration setup where the team definition applied. In the case of translation via LSPs, developers and localisers are usually physically and organisationally separated, although sometimes it happens that localisers are sent to work on location at a client's organisation. Nonetheless, as discussed previously, developers and localisers do not share success criteria or responsibilities. Yet, they still form a *work system*, a social unit on a project.

4.2.3.2.3 Cross-Disciplinary Knowledge

Developers and localisers attempt to educate their collaborators about those parts of their own jobs that they feel are relevant. For example, some developers proactively write comprehensive explanations of placeholders and control characters for localisers on their own initiative (DM7). Localisers trying to educate developers without a particular triggering event, e.g. a specific localisation issue, are significantly rarer.

It also appears that knowledge proliferation attempts are often foiled by framing the subject based on one discipline. For example, despite the explanation on placeholders, the developer did not provide context information for the translation. One might presume that localisers understanding placeholder syntax makes more sense for developers than localisers needing context information for translations.

Professionals with knowledge of both development and localisation appear to be very valuable for an organisation, yet do not always receive the acknowledgement they deserve. A number of localisation research papers mention localisation engineers (e.g. Wasala *et al.*, 2012; Anastasiou, 2009), but not many localisation engineers participated in the interviews. It appears that LSPs prefer to hire project managers with a translation

background, and software companies who have the foresight to assign an engineer to localisation usually do not blindly follow the agency model, but either employ translators or manage freelance translators themselves.

Ironically, interviewees which appear to fit the job title of localisation engineer report that they at times encounter the same incomprehension from their developer colleagues that otherwise is experienced by localisers. For example, colleagues focused on their engineering tasks and still failed to provide context information. This provokes the question whether an interdisciplinary barrier is a barrier between representatives of two disciplines, i.e. developers and localisers, or between the two subjects itself.

4.3 Discussion

The results of the GT analysis were presented in the shape of a theory of interdisciplinary collaboration in software localisation, which can be seen as a model to explain the behaviour reported in the accounts of interviewees. This model explains what happens during the collaboration of developers and localisers. A number of models have been proposed to describe the behaviour of individuals and organisations in business settings. Identifying existing models fitting to the data will conclude the presentation of qualitative results.

4.3.1 Borrowing of Models and Concepts across Disciplines

A frequent technique of interdisciplinary collaboration is “borrowing for instrumental purposes” (Klein, 1990, p.86), yet cross-disciplinary borrowing can be problematic: borrowed material may be misunderstood and distorted, it might be used out of context, or it might be controversial or abandoned in the source discipline – borrowers accordingly bear the “burden of comprehension” to obtain a basic understanding of whatever model is borrowed (Klein et al., 1990, p.88). Failure to do so can lead to considerable problems, as observed in this research's data.

An immediate reflex might be to look for instances where development borrows models and concepts from localisation or translation studies, but it appears that this happens rarely, and there are no applicable models of translation studies for developers to borrow from. However, it was observed that localisation, and there in particular tools and processes, borrow models from software engineering and development.

4.3.2 Interdisciplinary Work as a Social System

In existing research, the collaboration of workers from different disciplines has been examined in the context of team work. Colloquially, team work seems to be understood as any number of people working as a unit, but most publications imply a stricter definition. People merely working together form a group, whereas a team has a common goal and has usually been selected specifically to combine certain skills, but also for strategic purposes such as representing all departments of a company in order to obtain overall commitment (Maylor, 2010). Further, teams are generally assumed to work in the same physical space, a central ingredient for forming group commitment and cohesion, and have the authority to manage their own work.

In project management, this concept has been proliferated as cross-disciplinary team work. Despite the understanding of team as an organisational unit, they can include members from outside the organisation such as customers or providers. Social factors of cohesion, group conformance and group pressure are intended to increase team efficiency and effectiveness of individuals. Interplay between the social component of collaboration and cross-functional cooperation outcome forms a feedback loop as goals, rules, procedures, and accessibility are influenced by the former and determine the latter (Pinto, 2015). Cross-functional teams often aim to optimise solutions by comparing different views of multiple disciplines on the same problem. In localisation the disciplines complement each other, but don't collaborate on the same specific work steps as such. In fact, the primary mode observed in this research was not that of developers and localisers working as organisational unit, but rather the control of the localisation process by software developers, similar as discussed by Cooper (2004, p.207). This often went hand in hand with minimal communication, although communication has been identified as a crucial method to combat the often paradoxical and contradicting nature of interdisciplinary collaboration (Donnellon, 1993).

Some of the aspects of cross-disciplinary team work apply to the collaboration in software localisation, e.g. the combination of translation and software engineering skills. On the other hand, the implied physical access is mostly not realised due to outsourcing. Stoeller (2011) refers to this as virtual team, although this might be a misnomer as virtual teams in the scientific literature never account for the sometimes total lack of communication

encountered in the interview accounts of localisers and developers. And even in virtual teams, it seems critical to uphold team cohesion in order to access the cognitive processes associated with being the member of a group (Ale Ebrahim *et al.*, 2009). Even more critically, localisers often have no input into the management of their work.

A theory of interdisciplinary collaboration that does not rely on the assumption of a team structure is that proposed by Sverre Sjölander (in Klein, 1990, p.71), who has proposed ten stages of interdisciplinary collaboration. Initially, colleagues from different areas stick to their own discipline (stage 1) and colleagues from other disciplines are seen as having nothing to contribute to the project (stage 2). First interdisciplinary discussions are exceedingly abstract, increasing the chance of first agreements on very basic levels such as an agreement that product quality is important (stage 3). Then, the colleagues start forming a common vocabulary (stage 4), followed by first successful concrete discussions (stage 5) and a further construction of interdisciplinary jargon (stage 6), interrupted by frustration and failure due to the complexity of communication and the task (stage 7). Only when this stage is overcome can collaborators start seeing beyond their own discipline (stage 8), engage with different disciplines (stage 9), and finally become true interdisciplinary collaborators by examining their common task from all possible angles without blinders (stage 10).

A group of co-workers can get stuck in any of the mentioned phases. Sjölander's model emphasises social aspects of interdisciplinarity that were found in the interview accounts of developers and localisers, such as communication, negotiation and development of a common jargon. Where this fails, participants instead remain focused on their own discipline, and what happens is what O'Donnell *et al.* (1997) call *multidisciplinary group work* instead of interdisciplinary collaboration. This, too, was observed during this research. A particular observation might relate to the hierarchy implied within a perception that another discipline is not contributing, to be discussed next.

4.3.3 Dominance of Software Engineering

The relationship between software engineering and other disciplines in software development has been identified as "interdisciplinary challenges of software practice" (Andelfinger, 2002, p.200). Hirschheim and Klein (1989, p.1212) conclude that "information systems development approaches are influenced by assumptions from more

than one paradigm. However, the influence from one paradigm is typically dominant." Software engineers impose their understanding of a different discipline's subject matter when trying to tackle it in software. In doing this, they are prone to overestimating the practical use of the deterministic paradigm, mathematics and logic, and conceptual notions such as users, role and processes, and apply their own paradigm on collaborating disciplines regardless of fit (Low *et al.*, 1996; Green, 1994; Fetzer, 1988; Leith, 1986). This leads to a focus on technical aspects at the disadvantage of other aspects. Software engineers "see the world in terms of a computational model and fail to stand outside that model" (Leith, 1986, p.552). In fact, there is even a certain internal rivalry between software development's three sub-disciplines computer science, software engineering and information systems (Glass *et al.*, 2004).

Evidence of a dominance of software engineering was found in the interview data, specifically in statements implying that technological solutions to handle different cultural expectations are sufficient representations of culture itself, or the notion that translation is equivalent to a mathematical mapping.

This dominance of engineering in software development has been examined repeatedly in the context of HCI and UX design. Illmensee and Muff (2009) conducted a study how agile development methods affected user-centred design (UCD) and UX design. UCD processes were observed to be too cumbersome and slow to fit to agile processes (Detweiler, 2007), and it was concluded UCD and UX need to adapt to engineering. Abdelnour-Nocera *et al.* (2007) found that the quickly iterating process can make it difficult for developers to consider user needs.

In a practical case study, Sy (2007) contrasted UCD processes in waterfall and agile environments. A particular problem was that iterations were too short to prepare a design, conduct testing, and provide results in time. Sy proposes staggering UCD tasks across multiple iterations so that through advance planning, testing of a design skips one iteration. The study is an illustrative example of the influence software development models have on related non-engineering processes because it illustrates the wide range of consequences and necessary adaptations following a change from linear to iterative development.

Zhang *et al.* (2003) examined the role of HCI design in software development and concluded that HCI aspects are not considered sufficiently in development models because these are based on organisational needs, not human needs, along with a misconception that HCI only concerns the visible interface design such as screen and menu layout, colour choice and icon design.

But accessibility, usability and hedonistic enjoyment of software sit at the boundary between social aspects of computing and software development and are quite different from software engineering (Low *et al.*, 1996): software engineers prefer to tackle hard problems (Robinson *et al.*, 1998), but UX matters are soft problems. Software engineering is based on finding commonalities by ignoring differences, whereas UX is based on ignoring commonalities and examining differences towards improvement (Christiansen, 2010).

Accordingly, there have been calls to further integrate UX and HCI development into the overall development process (e.g. Zhang *et al.*, 2003; Maxwell, 2002), just as other activities which are used during software development (Kruchten, 2005; Bunting *et al.*, 2002). These mirror similar calls mentioned in chapter 2 regarding an integration of localisation into software development, reported integration problems of non-engineering processes (e.g. Smith *et al.*, 2004), and calls by Andelfinger (2002) to integrate disciplinary, methodological and procedural practice of other disciplines into software development. This aligns to the research results of Kim and Kang (2008) who surveyed 243 managers of cross-functional teams and identified trust, cohesion, and an alignment of goals, visions and professional culture as success factors. Randall *et al.* (1993) have equally suggested an examination of interdisciplinary work during systems design, but considered the interdisciplinary gap between software engineer and user.

Does that mean that dominance of software engineering extends to the development of international software, including localisation? Klein (2005, p.44) asserts that in interdisciplinary collaboration, “status hierarchies and hidden agendas will […] interfere”, and indeed localiser interviewees certainly report a feeling of little control over the quality of their work, and Pym (2008) refers to localisers being in a servile position. As discussed, many accounts imply a tendency to consider translation a technical concept

and localisation a larger feature to be implemented through technology. Insofar, localisation encounters the same problems as UX design.

One of the ways in which the dominance establishes itself is via a locus of power. The interview accounts show that internationalisation choices, e.g. if and how to internationalise, is made by developers due to their ownership of the source code. Organisational decisions are made not necessarily by developers themselves, but certainly on the developer side. Further, localisers depend on context information and of course sources from developers in order to localise software.

The results show not only the apparent hierarchy here, they also illustrate different consequences depending on how developers act, for example when choosing outsourcing or refusing to provide additional information.

4.3.4 Authority and Hierarchy

French and Raven (1959) define five types of power: legitimate power derived from a hierarchy, reward power obtained through the ability to reward, coercive power obtained from the ability to punish or use force, expert power obtained through specialist skills or knowledge, and referent power obtained through social support.

If one accepts that the customer-provider relationship between developers and localisers might create a hierarchy, then the power that developers hold over localisers would be legitimate power. Otherwise, since customers generally reward as similar to that between a customer and a provider, then this might be a relationship of reward power. However, assuming that the previously discussed dominance of software engineering is at work, a coercive power relationship is indicated. In each case, developers drive activities of localisers that are understood to be less than optimal for localisation.

A dominance of software engineering might explain the underappreciation of linguistic processes or that requirements of linguistic quality might not receive as much concern as needed in a larger software development process. However, the interview results also showed localisers apparently conforming to this hierarchy, e.g. by silently accepting a lack of context information or engaging in an organisational structure detrimental to the quality of their work. These cannot be explained by a power advantage on the side of developers alone.

Research exists on the potential effects of hierarchy and authority on behaviour. Arguably the two most prominent might be the agentic state theory and the authority gradient.

The agentic state theory was developed to explain results of Stanley Milgram's experiment on obedience to authority figures (Milgram, 1963), in which participants were led to act immorally through nothing else than perceived authority. The agentic state theory says that people may enter an agentic state, i.e. act on another person's will or as another person's agent, if that other person is seen as legitimate authority that will take responsibility for the action (Milgram, 1974).

Authority gradients have been used to explain observed behaviour in the context of medical operations and airplane operations (Hagen, 2013; White, 2012; Cosby and Croskerry, 2004), when operating team members and crew accepted errors of authority figures or superiors without contradiction despite discerning the error, sometimes with fatal consequences for others or even themselves. The authority gradient refers to a lack of communication of hierarchically lower team members in order to avoid being in contradiction to authority.

Both agentic state theory and authority gradient have been developed in much more severe and critical contexts than the development of international software generally is, and with arguably much more stress for the hierarchically lower individual than was apparent from the interview accounts.

Yet both theories fit interview accounts. Some accounts suggest that in an outsourcing setup, localisers might perceive a hierarchy gradient in the customer-provider relationship and thus an authority relationship in the sense of the agentic state theory. While Milgram's original experiment bears no resemblance to what happens during localisation, the agentic state theory merely defines conditions under which an individual will act unconditionally on another person's orders, which fits to e.g. a localiser following unsuitable processes. Further, it is reasonable for localisers to think that since the unsuitable process is the choice of developers, they will also take responsibility for consequences such as detrimental quality.

However, there are problems. The agentic state theory seems to have been derived inductively, and its fit to actual observations is a matter of debate (Nissani, 1990).

Further, the agentic state theory explains conformity behaviours merely through authority.

The authority gradient, on the other hand, is an explanation of obedience in a professional work relationship and based on behaviour observed in practice. Developers are still the erring authority and remain undisputed by the specialist in a hierarchically lower position who wants to save his status. In the interview accounts, the status would be the business relationship between developer and localiser. While not being a perfect fit as localisation issues seem to be blamed on the localisers after all, it arguably aligns with the passivity of a customer-is-king or work-to-rule attitude noted in a number of accounts.

However, there is yet another theory that might explain the observed social dynamics while completely doing away with the assumption of authority or dominance.

4.3.5 The Theory of Agency

The theory of agency, also referred to as principal-agent relationship, refers to the behaviour of agents and principals with different interests and asymmetric knowledge. Principals employ agents to act on their behalf, but the agent can use his advantage in knowledge or information to keep the principal in the dark (Jensen and Meckling, 1976). This theory has been widely applied in many economic contexts. In software development, it has been applied to explain loss of quality through developers taking shortcuts in order to alleviate schedule pressure (Austin, 2001) and to develop incentive schemes to overcome unaligned goals of software companies and its employees (Banker and Kemerer, 1992).

Primary conditions of a principal-agent relationship are differing incentives for agent and principal and an information advantage of the agent. The development side is the principal, who is a customer of a localisation side, i.e. a freelance translator or LSP, in the role of an agent. The development side is incentivised by paying the localisation side for a good localisation. In some instances of principal-agent relationship, the agent is incentivised to keep profit up. Here, localisers are incentivised to keep cost and effort down and fulfilling the developer's expectation not to be bothered. This is achieved by not conducting research and not asking for additional information. The information

advantage of localisers towards developers is the developers' inability to comprehensively test localisation quality, understand translation complexities or evaluate localiser activities. The general reduction of individual effort in a group setting is called *social loafing* (Latane *et al.*, 1979). The specific dynamic suspected here has been referred to by Alchian and Demsetz (1972) as shirking in group work enabled by the disproportionate cost of metering each group member's performance. And indeed, it has been established that localisation testing is effortful. It seems to align with Durkheim's (1893, in Haralambos *et al.*, 2004) *organic solidarity* in an industrial society where specialisation has rendered social solidarity based on similarity ineffective. In such a society, self-interests of individuals need to be regulated by rules and contracts.

4.3.6 Organisational Control in Software Localisation

Initially, the terms localiser and developer in this research referred to individuals. However, as this present discussion progresses, they have come to include software companies and LSPs. It sets a different context for an understanding of the potential difficulties observed in the agency model. LSPs are not linked to a specific project in the same way software companies are. They have little attachment, and move from one project to the next with little guarantee of continuity. And if it is the customer's expectation to 'protect' developers from interacting with translators, then an LSP will inadvertently act as a filter, despite better knowledge and with all the conceivable limitations this brings with it. LSPs and freelancers judge their work as vendors aiming at customers' contentedness. In that function, actual localisation quality is a secondary priority. The strict separation of translation and development leads to localisation issues. This could be resolved easily by reducing the separation. So why is this insisted upon?

During research of the Tavistock Institute, it was observed that in two groups of miners, productivity of the social group, where workers knew each other across shifts, was 25% higher and absence from work 60% lower compared to a group where workers did not know each other and work had been partly automated. The difference was explained through a lack of trust which affects the work habits of people who know they work in a highly risky business and whose wellbeing is dependent on the conscientiousness of their co-workers (Trist and Murray, 1990). So, once miners did not know their colleagues from other shifts anymore, they double-checked work done in previous shifts before working

themselves in order to ensure their own safety. Bafflingly, it was found that neither management nor unions were interested in the superior aspects of social, autonomous groups with less automation, concluding that the *partialised system*, as it was called, sacrifices efficiency for control and power.

Similarly, it has been suggested that automation and computer usage at the workplace leads to the loss of skills, so-called *de-skilling*, on the side of workers (Smith, 2013; Kling, 1996a, 1996b). Braverman (1999) and Zuboff (1988) have suggested that this is a deliberate attempt by management to minimise work of skilled workers and strengthen existing hierarchies in an implementation of scientific management: as much work-specific knowledge as possible is moved into the organisation, its processes and tools, so that eventually the workers are dependent on it for work.

There is no quantitative evidence how much a more inclusive approach to translation would improve efficiency. There are interview statements suggesting that translation would be faster and more efficient if the separation between translators and engineers were less strict and there would be less bureaucracy. However, some accounts might also be interpreted to mean that the disadvantages of the LSP model, i.e. strict separation and automation, are deliberately ignored. So, the findings of this research fit into the pattern already discovered in the Tavistock studies, and it can be hypothesized whether their conclusions, i.e. a drive for management control overriding operational efficiency, holds in software localisation as well.

In fact, it is not hard to imagine that CAT and MT technologies take over more and more work aspects in translation and lead to a de-skilling of translators, as argued by Séguinot (2007) and reported by translators themselves (LeBlanc, 2013). The controlling effect of mechanisation, automation and in this context also information technology has been widely discussed (e.g. Orlikowski, 1991; Browne, 2005). Even software engineering has been suggested to be subject to de-skilling as a consequence of division of labour and increasing use of programming aids that fragment the work, e.g. structured or object-oriented programming (Glass, 2005; Friedman, 1993).

4.4 Summary

This chapter presented and discussed the findings from the qualitative part of our research. Based on interview accounts, a grounded theory of interdisciplinary collaboration in software localisation was created. Interview accounts and the theory were discussed in the light of literature on software localisation and software development previously discussed in the literature review, and additional research and literature on organisational relationships and the sociology of work. Those findings allow return to those research questions prompting the interviews.

RQ 1 asked how localisation is conducted individually and collaboratively, and what shapes the activities of developers and localisers. Analysis of the interview accounts show a broad range of activities and processes employed to localise software. Besides their main activities, e.g. programming or translating, interviewees' activities with regards to localisation were focused on integrating their work, including the required input and the resulting deliverables, into the overall software development by organising or facilitating the collaboration with the respective other discipline. In this, the work of developers and localisers is strongly influenced by the strategic choice of conducting localisation in-house or out-of-house. Activities are shaped by external influences such as success criteria, limitations and affordances of tools, and limitations and tasks specified in the processes prescribed in the overall organisation. Additionally, the activities are modified to avoid or handle conflicts, i.e. failures and impasses, previously experienced or expected in localisation.

RQ 2 asked how localisation issues are caused during localisation and internationalisation. According to the theory generated from the interview accounts, localisation issues can be results of the hierarchical relationship between developers and localisers and potentially unaligned goals. Developers, as guardians of the code and possible customers of localisation in an outsourcing model, enjoy a privileged position compared to localisers, extending to the developers' priorities. As a consequence, their relationship with localisers can easily develop towards a dysfunctional regime in which processes and tools cater more and more for developers and development, and ever less for localisers and localisation. The more dysfunctional the relationship is, the less localisers request necessary information from developers, notify developers of potential issues, or

eventually focus on the localisation projects' goals, and the more they shift their work and activities towards their unique interests, and *vice versa*. Eventually, developers and localisers settle into a relationship in which localisation factors have been superseded by alternative interests of the collaborating professionals. Those goals of software localisation which are not among their priorities are compromised and cost, quality or schedule issues occur.

Chapter 5 Quantitative Results

In the introduction, it was found that an examination of the distinctness of developers and localisers (RQ3) and dependencies between localisation effort and properties of development projects (RQ4) lends itself to the use of quantitative methods. Chapter 3 discussed a research method to examine these through a survey gathering biographical data of participants, project-related information, and instruments measuring cultural competence, attitude towards localisation, self-efficacy in localisation, and localisation effort. Hypotheses, survey construction and analysis methods were detailed in subsections 3.3.2 and 3.3.4. This chapter will present the results of the statistical analysis.

5.1 Sample Description

Of 301 respondents who started the survey, 175 did not complete it. As a convenience sample was used, both non-responses and non-completers are anonymous so it was not possible to contact them about their reasons for cancelling. Of the 126 survey completions, 6 were ruled out because the respondents had no experience in the development of localised software. This left a total of 120 survey submissions for analysis.

The sample is sufficient with regards to the intended statistical tests. The balanced role distribution allows comparison of information from respondents in technical roles to those in non-technical roles. Equally, respondents' native languages are balanced enough in the sample to allow comparisons between native English and non-native English respondents. There is no perceivable skew that would invalidate a cross-sectional study of continuous variables such as cultural competence and attitude towards localisation.

Beyond this, biographical details suggest that the sample is generally not homogeneous. There is, however, a clear limitation regarding nationality: most respondents are either German, British, or US-American. Accordingly, the sample exhibits a clear skew towards the German and English language. Despite major offshoring destinations such as India virtually not being represented in the survey, the results should still be representative since Europe and the USA are among the dominant industry players in software development and IT spending, and house the top 25 software companies (United Nations, 2012), hence it must be considered that the majority of development and design decisions are made in these nations.

5.1.1 Respondents

The average age of respondents was 38.4 years, with the youngest respondent being 17 and the oldest 63 years old. 89 respondents (74%) were male. On average, respondents had been working on international software for 9.0 years, ranging from 1 year to 35 years. Most respondents were German (27%), followed by British (19%) and US-American (10%). Three respondents (2.5%) did not identify a specific nationality. The remaining respondents were from 24 other countries.

Table 5-1 Nationality of respondents

Country	Frequency	Percent
Argentinian	1	0.8
Austrian	2	1.7
Belgian	2	1.7
Brazilian	3	2.5
British	23	19.2
Canadian	1	0.8
Chinese	3	2.5
Danish	2	1.7
Dutch	2	1.7
English	2	1.7
Finnish	2	1.7
French	3	2.5
German	32	26.7
Hungarian	1	0.8
Indian	4	3.3
Indonesian	1	0.8
Irish	2	1.7
Italian	2	1.7
Malaysian	1	0.8
Nepalese	1	0.8
Palestinian	1	0.8
Polish	4	3.3
Russian	2	1.7
Spanish	5	4.2
Swedish	1	0.8
Tunisian	2	1.7
unknown	3	2.5
US-American	12	10.0

The most common highest education level was a master's degree or equivalent (53%), followed by a bachelor's degree or equivalent (30%), a doctoral degree or equivalent (10%), and a high school degree or equivalent (8%). For the roles, multiple responses were possible. 37% of the respondents described their usual roles in software development as software engineer, 24% as user interface designer, 27% as software architect, 10% as business analyst, 38% as project manager, 32% as translator/localizer, and 10% as technical editor. Overall, 70 participants were grouped as developers and 50 as localisers.

Table 5-2 Highest level of education of respondents

Degree	Frequency	Percent
High School, Grammar school or equivalent	9	7.6
Bachelor's degree or equivalent	35	29.7
Master's degree or equivalent	62	52.5
Doctoral degree or equivalent	12	10.2

Table 5-3 Role of respondents

Role	N	Percent	Percent of cases
Software engineer	43	20.8	37.4
User interface designer	28	13.5	24.3
Software architect	31	15.0	27.0
Business analyst	12	5.8	10.4
Project manager	44	21.3	38.3
Translator/localiser	37	17.9	32.2
Technical editor	12	5.8	10.4
Total	207	100.0	180.0

Table 5-4 Localisation training of survey respondents

Knowledge source	N	Percent	Percent of cases
Literature about localization	56	26.4	46.7
Literature partly about localization	49	23.1	40.8
Informal training	62	29.2	51.7
Formal training	22	10.4	18.3
None of the above	23	10.8	19.2
Total	212	100.0	176.7

18% of all respondents had received formal training about software localisation, compared to 19% without any training. This, however, strongly depended on role. When considering only respondents in a development role, the numbers for formal training decreased by almost half to 6% and the numbers for respondents with none of the listed educational measures increased by almost half to 15%, whereas the numbers for the remaining options stayed within 10% of their value for all respondents. Most of the formal training was received by translators.

5.1.2 Projects of International Software

Many items asked participants about properties of their most recent localised software project. 71% of these projects were application software, 33% were websites, 19% were system software, 16% were mobile applications, 9% were video games, and 7.5% were firmware. Typical users of these projects, as reported by the respondents, were companies (56%), followed by private end-users (54%), educational institutions (22%), government institutions (20%), software developers (15%), and finally scientists (13%).

Table 5-5 Software types of reported projects

Software type	N	Percent	Percent of cases
Application	85	45.5	70.8
Video Game	11	5.9	9.2
Website	40	21.4	33.3
Mobile App	19	10.2	15.8
System Software	23	12.3	19.2
Firmware	9	4.8	7.5
Total	187	100.0	155.8

Table 5-6 User types of reported projects

User type	N	Percent	Percent of cases
Private end-users	65	30.1	54.2
Software developers	18	8.3	15.0
Scientists	16	7.4	13.3
Companies	67	31.0	55.8
Government institutions	24	11.1	20.0
Educational institutions	26	12.0	21.7
Total	216	100.0	180.0

Of interest were also the frequencies of what parts of software were localised. In almost all projects (95%), the user interface text was localised. About two thirds of all projects also localised data formatting (68%), and units of measurements (63%). Other forms of localisation are much rarer, with about a quarter of all projects localising functionality (29%), navigation and layout (26%), followed by colours, images and sounds (19%), and feature sets (18%).

Table 5-7 Localised software elements of reported projects

Localised software elements	N	Percent	Percent of cases
User interface text	114	29.8	95.0
Formatting, e.g. time and date and sort orders	82	21.5	68.3
Units, e.g. measurements, currency and paper sizes	75	19.6	62.5
Colours, graphics and sound	23	6.0	19.2
Navigation and layout	31	8.1	25.8
Functionality	35	9.2	29.2
Feature sets	21	5.5	17.5
Unknown	1	0.3	0.8
Total	382	100.0	318.3

The vast majority of projects, 94 (80%), were commercial, compared to 21 (18%) non-commercial projects and 3 (3%) projects of unknown commerciality. 54 projects (45%) were localised into 1-5 languages, followed by 27 projects (23%) into 6 – 15 languages, 21 projects (18%) projects into 16 – 30 languages, and 17 projects (14%) into more than 30 languages.

Table 5-8 Number of languages of reported projects

# of languages	Frequency	Percent
1 - 5	54	45.0
6 - 15	27	22.5
16 - 30	21	17.5
More than 30	17	14.2
Unknown	1	0.8
Total	120	100.0

44% of the projects followed an agile approach, 15% of the projects followed a waterfall approach, and 3% followed a spiral model approach. 15% of the projects followed no development model.

Table 5-9 Development model of reported projects

Development model	Frequency	Percent
Waterfall model	19	15.8
Spiral model	3	2.5
Agile model	53	44.2
No particular model	18	15.0
Unknown	27	22.5
Total	120	100.0

Translations were provided for the projects by customers (9%), machine translation (13%), crowdsourcing (16%), employees in a non-translation role (24%), employed translators (34%), freelance translators (41%), and translation agencies (43%). 3% of respondents did not know how translations for their projects were provided.

For the transmission of translations between translators and developers, projects used random file formats (4%), XLIFF (15%), self-developed formats (18%), program files (20%), XML (24%), standard desktop formats including Excel (27%), and online databases, TMs or CMS (36%). 9% of respondents did not know how translations were transported.

5.2 Variable Distributions and Data Preparation

Prior to analysis, the data was screened for false or inappropriate data such as monotonous replies indicating a click-through rather than thoughtful participation, and inexplicable outliers. Data screening showed that one respondent entered a nonsensical experience length in software localisation, two had not clearly identified their highest education level, and three had not clearly identified their nationality. Three respondents had answered all CQ-related items monotonously, two had not specified commerciality of their project, one did not know what elements were localised, 27 did not know the development model of their project, and one did not know the number of target locales.

In order to select the correct statistical tests, the distribution of tested variables needs to be known. Specifically, the normality or non-normality of a variable's distribution determines the suitability of different correlation tests. For example, the Pearson test to test the correlation of two continuous variables requires both variables to follow a normal distribution. Otherwise, the Spearman rank correlation is to be used (Bryman and Cramer,

1995). Because of the sample size of n < 2000, the Shapiro-Wilk test was used to test for normality (Norušis, 2006). Table 5-10 shows the distributions of the continuous variables.

Table 5-10 Variable distributions

Construct	Shapiro-Wilk sig. (p)	Distribution
Cultural Competence	.01	not normal
Metacognitive Cultural Competence	.00	not normal
Cognitive Cultural Competence	.08	normal
Motivational Cultural Competence	.00	not normal
Behavioural Cultural Competence	.00	not normal
Attitude Towards Localisation	.00	not normal
Localisation Effort	.02	not normal

To see whether non-native English speakers score higher on cultural competence than native English speakers, the mean score of both groups was compared. English as native language was inferred from nationality: participating nationals of a country in which English is the first language, e.g. the UK, USA, Canada, Australia, New Zealand, or South Africa, were sorted into the group of native English speakers. Participating nationals from any other country were sorted into the group of non-native English speakers. Three respondents' completed surveys were not considered for this test as they had not disclosed their nationality in the survey.

5.3 Hypothesis Results

An overview of all tested hypotheses and data exclusions is shown in Table 5-11. The significance threshold for statistical significant was chosen as p < .05.

Table 5-11 Overview of the survey analysis results

ID	Hypothesis	Test	p	Result	n	Exclusion reason
H1	Developers score lower than localisers on CQ	T-Test	.00		116	monotonous CQ
H1a	… on metacognitive CQ	T-Test	.01		116	monotonous CQ
H1b	... on cognitive CQ	Pearson	.00		116	monotonous CQ
H1c	… on motivational CQ	T-Test	.12	rejected	116	monotonous CQ
H1d	… on behavioural CQ	T-Test	.09	rejected	116	monotonous CQ
H2	Developers & localisers assume different loc.scope					
H2a	… for UI text	Phi coefficient	.43	rejected	119	role not clear
H2b	… for formatting	Phi coefficient	.70	rejected	119	role not clear

H2c	… for units	Phi coefficient	.70	rejected	119	role not clear
H2d	… for colours, sound etc.	Phi coefficient	.09	rejected	119	role not clear
H2e	… for navigation	Phi coefficient	.11	rejected	119	role not clear
H2f	… for functionality	Phi coefficient	.36	rejected	119	role not clear
H2g	… for feature sets	Phi coefficient	.39	rejected	119	role not clear
H2h	Developers assume a smaller localisation scope than localisers	T-Test	.27	rejected	119	role not clear
H3	Developers score lower on ATL than localisers	T-Test	.01		119	role not clear
H4	Developers assume less responsibility for localisation than localisers	Phi coefficient	.63	rejected	119	role not clear
H5	Developers have a higher SEL than localisers	T-Test	.52	rejected	119	role not clear
H5a	Developers have a higher SEU than localisers	T-Test	.04		119	role not clear
H5b	SEL is correlated with SEU	Spearman	.00		120	
H6	Cost, quality and time priorities differ between developers and localisers	Chi-square	.43	rejected	119	role not clear
H7	Software success factor priorities differ between developers and localisers					
H7a	… on maintainability	T-Test	.02		119	role not clear
H7b	… on reliability	T-Test	.12	rejected	119	role not clear
H7c	… on correctness	T-Test	.56	rejected	119	role not clear
H7d	… on execution Speed	T-Test	.09	rejected	119	role not clear
H7e	… on usability	T-Test	.09	rejected	119	role not clear
H7f	… on power	T-Test	.01		119	role not clear
H7g	… on popularity	T-Test	.04		119	role not clear
H7h	… on financial success	T-Test	.29	rejected	119	role not clear
H8	Localisation training is correlated with CQ	Spearman	.00		116	monotonous CQ
H8a	… with metacognitive CQ	Spearman	.00		116	monotonous CQ
H8b	... with cognitive CQ	Spearman	.01		116	monotonous CQ
H8c	… with motivational CQ	Spearman	.01		116	monotonous CQ
H8d	… with behavioural CQ	Spearman	.00		116	monotonous CQ
H8e	For developers, loc.training is correlated with CQ	Spearman	.03		70	developers only
H8f	… with metacognitive CQ	Spearman	.02		70	developers only
H8g	... with cognitive CQ	Spearman	.17	rejected	70	developers only
H8h	… with motivational CQ	Spearman	.06	rejected	70	developers only
H8i	… with behavioural CQ	Spearman	.00		70	developers only

H9	Native English speakers score lower than non-native Engl. speakers on CQ	T-Test	.70	rejected	113	3 nationality unclear; 4 monotonous CQ
H9a	… on metacognitive CQ	T-Test	.66	rejected	113	see above
H9b	… on cognitive CQ	T-Test	.60	rejected	113	see above
H9c	… on motivational CQ	T-Test	.92	rejected	113	see above
H9d	… on behavioural CQ	T-Test	.56	rejected	113	see above
H9e	Native German speakers score lower on CQ than non-German speakers	T-Test	.93	rejected	113	see above
H10	LE is affected by s/w type…					
H10a	… application	T-Test	.24	rejected	120	
H10b	… video game	T-Test	.00		120	
H10c	… website	T-Test	.82	rejected	120	
H10d	… mobile app	T-Test	.46	rejected	120	
H10e	… system software	T-Test	.03		120	
H10f	… firmware	T-Test	.23	rejected	120	
H11	LE is affected by user type…					
H11a	… private end users	T-Test	.18	rejected	120	
H11b	… software developers	T-Test	.22	rejected	120	
H11c	… scientists	T-Test	.24	rejected	120	
H11d	… companies	T-Test	.43	rejected	120	
H11e	… government institutions	T-Test	.36	rejected	120	
H11f	… educational institutions	T-Test	.26	rejected	120	
H12	LE is affected by customer-user identity	ANOVA	.83	rejected	109	customer-user identity not clear
H13	LE is affected by number of target languages	ANOVA	.04		119	# of languages unknown
H14	LE is affected by development model	ANOVA	.00		90	3 spiral model; 27 model unkn.
H15	LE is affected by project commerciality	ANOVA	.15	rejected	114	commerciality not known
H16	ATL is correlated with CQ	Spearman	.00		116	monotonous CQ
H16a	For developers, ATL is correlated with CQ	Spearman	.06	rejected	81	developers only
H17	SEL is correlated with CQ	Spearman	.06	rejected	116	monotonous CQ
H18	SEL is correlated with ATL	Spearman	.00		120	

For each statistical test used, a separate table shows the tested hypotheses and result details results. Independent sample t-test results (Table 5-12) include mean (M) and standard deviation (SD) of each group G1 and G2, t statistic, degree of freedom df, and significance p. Pearson correlation test results (Table 5-13) include correlation coefficient

r and significance p. Chi-squared test results (Table 5-14) include Chi-square value, degrees of freedom df, and significance p. Phi coefficient test results (Table 5-15) include Phi value and significance p. Spearman rank correlation test results (Table 5-16) include correlation coefficient r and significance p. ANOVA results (Table 5-17) include degrees of freedom df, F value of the relationship between explained and unexplained variance, significance p, and for significant results effect size f as well as Tukey honest significant differences (HSD, Table 5-18 and Table 5-19).

Table 5-12 Independent samples t-test results

ID	Hypothesis	G1 M / SD	G2 M / SD	t	df	p
H1	Developers score lower than localisers on CQ	74.59 / 16.99	85.23 / 16.25	-3.31	114	.00
H1a	… on metacognitive CQ	17.01 / 4.30	19.14 / 3.33	-2.78	114	.01
H1c	… on motivational CQ	22.36 / 4.73	23.93 / 5.80	-1.59	114	.12
H1d	… on behavioural C	20.08 / 5.03	21.81 / 5.81	-1.69	114	.09
H2h	Developers assume a smaller loc. scope than localisers	4.46 / 1.72	4.82 / 1.77	-1.10	117	.27
H3	Developers score lower on ATL than localisers	42.43 / 7.01	45.80 / 6.50	-2.61	117	.01
H5	Developers have a higher SEL than localisers	9.59 / 2.51	9.24 / 3.30	.65	117	.52
H5a	Developers have a higher SEU than localisers	10.26 / 2.88	9.18 / 2.54	2.07	117	.04
H7	Software success factor priorities differ between developers and localisers					
H7a	… on maintainability	4.41 / 1.87	5.20 / 1.50	-2.41	117	.02
H7b	… on reliability	2.76 / 1.49	2.36 / 1.15	1.55	117	.12
H7c	… on correctness	3.05 / 1.63	2.34 / 1.81	-.59	117	.56
H7d	… on execution speed	5.22 / 1.63	4.71 / 1.39	1.73	117	.09
H7e	… on usability	2.47 / 1.48	2.00 / 1.40	1.72	117	.09
H7f	… on power	6.54 / 1.36	5.80 / 1.77	2.57	117	.01
H7g	… on popularity	6.22 / 1.86	6.87 / 1.34	-2.04	117	.04
H7h	… on financial success	5.34 / 2.50	5.82 / 2.19	1.08	117	.29
H9	Native English speakers score lower than non-native English speakers on CQ	77.56 / 19.15	78.92 / 16.81	-.39	111	.70
H9a	… on metacognitive CQ	17.56 / 4.80	17.92 / 3.75	-.44	111	.66
H9b	… on cognitive CQ	19.43 / 6.58	20.16 / 7.07	-.53	111	.60
H9c	… on motivational CQ	23.03 / 5.75	22.92 / 5.04	.10	111	.92
H9d	… on behavioural CQ	21.19 / 5.62	20.55 / 5.37	.59	111	.56

H9e	Native German speakers score lower on CQ than non-German speakers	78.23 / 17.33	78.58 / 17.68	-.09	111	.93
H10	LE is affected by s/w type…					
H10a	… application	34.29 / 7.56	31.97 / 10.37	1.20	49.55	.24
H10b	… video game	40.72 / 6.07	32.90 / 8.40	3.01	118	.00
H10c	… website	33.88 / 8.58	33.49 / 8.51	.24	118	.82
H10d	… mobile app	34.95 / 8.72	33.37 / 8.48	.74	118	.46
H10e	… system software	37.09 / 8.82	32.79 / 8.25	2.21	118	.03
H10f	… firmware	36.89 / 4.51	33.35 / 8.70	2.06	13.46	.23
H11	LE is affected by user type…					
H11a	… private end users	34.57 / 8.42	32.49 / 8.53	1.34	118	.18
H11b	… software developers	35.89 / 9.39	33.22 / 8.32	1.23	118	.22
H11c	… scientists	35.94 / 8.34	33.26 / 8.51	1.18	118	.24
H11d	… companies	34.16 / 8.06	32.92 / 9.05	.79	118	.43
H11e	… government institutions	35.04 / 8.71	33.26 / 8.45	.92	118	.36
H11f	… educational institutions	35.27 / 9.11	33.16 / 8.31	1.12	118	.26

As can be seen, it was found that developers score significantly lower on CQ and ATL than localisers, whereas the difference in SEL is not statistically significant. Developers prioritise maintainability and popularity higher than localisers, but prioritise application power lower. Developers and localisers reported no statistically significant differences for localisation responsibility. Prioritisation differences of reliability, correctness, execution speed, usability, and financial success were not statistically significant. Overall, developers and localisers should be assumed to prioritise the given software success factors the same. Similarly, the difference in project management priorities was also statistically not significant.

Table 5-13 Pearson test results

ID	Hypothesis	r	p
H1b	Developers score lower than localisers on cognitive CQ	-.34	.00

Table 5-14 Chi-square test results

ID	Hypothesis	Chi-square	df	p
H6	Project management priorities differ between developers and localisers	4.87	2	.43

Table 5-15 Phi coefficient test results

ID	Hypothesis	Phi	p
H2	Developers assume a different localisation scope than localisers		
H2a	… for UI text	-.07	.43
H2b	… for formatting	-.04	.70
H2c	… for units	-.04	.70
H2d	… for colours, graphics, sound	-.15	.09
H2e	… for navigation	-.15	.11
H2f	… for functionality	.08	.36
H2g	… for feature sets	-.08	.39
H4	Developers assume less responsibility for localisation than localisers	.05	.63

Table 5-16 Spearman rank correlation test results

ID	Hypothesis	r	p
H5b	SEL is correlated with SEU	.44	.00
H8	Localisation training is correlated with CQ	.29	.00
H8a	… with metacognitive CQ	.26	.00
H8b	… with cognitive CQ	.24	.01
H8c	… with motivational CQ	.23	.01
H8d	… with behavioural CQ	.31	.00
H8e	For developers, localisation training is correlated with CQ	.27	.03
H8f	… with metacognitive CQ	.28	.02
H8g	… with cognitive CQ	.17	.17
H8h	… with motivational CQ	.22	.06
H8i	… with behavioural CQ	.36	.00
H16	ATL is correlated with CQ	.29	.00
H16b	… with metacognitive CQ	.32	.00
H16c	… with cognitive CQ	.27	.01
H16d	… with motivational CQ	.21	.02
H16e	… with behavioural CQ	.21	.03
H16a	For developers, ATL is correlated with CQ	.22	.06
H16f	… with metacognitive CQ	.26	.03
H16g	… with cognitive CQ	.14	.22
H16h	… with motivational CQ	.13	.27
H16i	… with behavioural CQ	.11	.38
H17	SEL is correlated with CQ	.18	.06
H28	SEL is correlated with ATL	.42	.00

Table 5-17 ANOVA test results

ID	Hypothesis	df	F	Eta-squared	p
H12	LE is affected by customer-user identity	2, 106	.19	-	.83
H13	LE is affected by number of target languages	3, 115	2.78	0.07	.04
H14	LE is affected by development model	2, 87	7.03	0.14	.00
H15	LE is affected by project commerciality	1, 112	2.15	-	.15

Table 5-18 Post-Hoc Tukey HSD result for H13

# languages	# languages	MD	Std. Error	p	95% Confidence Interval	
					Lower Bound	Upper Bound
1 - 5	6 - 15	-.72222	1.96603	.983	-5.8477	4.4032
	16 - 30	-4.80688	2.14511	.118	-10.3992	.7854
	More than 30	-5.09259	2.31971	.131	-11.1401	.9549
6 - 15	1 - 5	.72222	1.96603	.983	-4.4032	5.8477
	16 - 30	-4.08466	2.42692	.337	-10.4116	2.2423
	More than 30	-4.37037	2.58254	.332	-11.1030	2.3623
16 - 30	1 - 5	4.80688	2.14511	.118	-.7854	10.3992
	6 - 15	4.08466	2.42692	.337	-2.2423	10.4116
	More than 30	-.28571	2.72135	1.000	-7.3803	6.8088
More than 30	1 - 5	5.09259	2.31971	.131	-.9549	11.1401
	6 - 15	4.37037	2.58254	.332	-2.3623	11.1030
	16 - 30	.28571	2.72135	1.000	-6.8088	7.3803

Table 5-19 Post-Hoc Tukey HSD result for H14

Development Model	Development Model	MD	Std. Error	p	95% Confidence Interval	
					Lower Bound	Upper Bound
Waterfall	Agile	-.92056	2.19742	.908	-6.1603	4.3191
	None	7.39181	2.70302	.020	.9465	13.8371
Agile	Waterfall	.92056	2.19742	.908	-4.3191	6.1603
	None	8.31237	2.24190	.001	2.9666	13.6581
None	Waterfall	-7.39181	2.70302	.020	-13.8371	-.9465
	Agile	-8.31237	2.24190	.001	-13.6581	-2.9666

Despite the confirmed difference in CQ, the localisation scope assessment is not statistically significant. In other words, while the observed difference between developers and localisers in CQ might affects software localisation, it does not affect both discipline's perception what aspects of software should be localised.

CQ is weakly correlated with training in localisation, suggesting that the training measures queried in the survey are somewhat effective. There is no statistically significant association between being a native English speaker and CQ. The data further shows that ATL is weakly correlated with CQ, but only for localisers, not for developers. Further, SEL is moderately correlated with ATL, but not with CQ.

There was no statistically significant difference in LE between commercial and non-commercial projects, nor was LE affected by type of user or whether the user was also a customer or not. A statistically significant increase of LE was observed for video games and system software, but not for the other software types. LE also increases moderately with number of target languages. Further, there is a large effect of software development model presence on LE.

5.4 Discussion

RQ 3 asked in what regards software developers and localisers are distinct. The results show that developers and localisers differ in cultural competence, attitude towards localisation, and self-efficacy in usability. They do not differ in self-efficacy in localisation, assumption of responsibility towards localisation, prioritisation of software success factors, and assessment of localisation scope.

RQ 4 asked what dependencies exist between localisation effort and properties of development projects. The results show that localisation effort increases for some types of software, with the number of target languages, and with the presence of a software development model. It remains unaffected by most software types, relationship to the user, or commerciality of a project.

The hypotheses confirmed and rejected by the data were suggested by existing literature on software localisation and are discussed in that context next.

5.4.1 Distinctness of Developers and Localisers

An influence of developer-localiser distinctness on cooperation has been discussed (e.g. O'Sullivan, 1989; Honkela *et al.*, 1997), particularly cross-disciplinary knowledge (Bauer and Rodrigo, 2004; Russo and Boor, 1993; Sikes, 2011; Immonen and Sajaniemi, 2003a) and disposition towards localisation (e.g. Sikes, 2011; Honkela *et al.*, 1997).

A number of differences could be confirmed. Developers have a lower cultural competence than localisers. However, on the other hand localisers do not have a lower self-efficacy in localisation than developers. The relationship localisers have with the discipline of software development is just as important as the one developers have with localisation (Law, 2003; Immonen and Sajaniemi, 2003a), and in this regard the lack of a difference in self-efficacy in localisation suggest that localisers understand the technical aspects of localisation just as well as developers. Overall, the results suggest that localisers seem to have more cross-disciplinary knowledge than developers. Interestingly, some of the localisers in the interviews seem to have been acutely aware of this advantage.

It has further been shown in the survey that developers are less positively predisposed towards localisation than localisers. However, developers and localisers exhibited only minor differences in assessment of software quality priorities, and no differences in project management priorities. While this suggests that there are no collaboration issues caused by success criteria in software development, e.g. cost and schedule over quality (see Boehm, 2011, 2006; Blackburn *et al.*, 1996), it leaves open the possibility that software success criteria supersede localisation success criteria, as suggested by Tuffley (2003) and Dunne (2011).

5.4.2 Cultural Competence and the Scope of Localisation

The research examined cultural competence of developers and localisers based on the assumption that developers need cultural awareness (e.g. Ryan *et al.*, 2009; Abufardeh and Magel, 2008a; Immonen and Sajaniemi, 2003a), for example to understand localisation requirements (Giammarresi, 2011; Kalliomäki *et al.*, 1997; Hoft, 1996). The survey results suggest that there are indeed differences in cultural competence between localisers and developers. Despite these, opinions regarding localisation scope were similar enough that they seem to be unaffected by this cultural competence gradient.

Regarding the sub-constructs of CQ, it is noteworthy that there are developer-localiser differences between metacognitive and cognitive CQ, but not between motivational and behavioural CQ. This is surprising as one would assume that localisers have more interest, and hence a higher intrinsic motivation, in learning about different cultures.

Results were not always as expected when examining factors that might influence cultural competence. While it is suggested that developers' cultural competence can be increased with training, this would nonetheless leave their attitude towards localisation unaffected. However, it is noteworthy that where Immonen and Sajaniemi (2003a, 2003b) found that few of their participants had read any books on software localisation, in this survey 43% of participants had reported that they had at least read a book on software localisation, and only 25% had not received any kind of formal or informal training.

Contrary to what is assumed (e.g. Carey, 1998), no statistically significant difference in the cultural competence or any of its sub-constructs between native and non-native English speakers could be found. There is room for error as the survey did not explicitly ask whether English was a native language. Instead, it asked for nationality, and English as native language was inferred from that information. It is conceivable that this procedure lead to incorrect assignments. Barring this confounding factor, it appears that cultural competence is not a matter of being multilingual. This interpretation is supported by findings of Khodadady and Shima (2012) who found no correlation between CQ and proficiency in English as a foreign language. Hence, adding professionals with a non-native English background would not automatically bring more cultural competence into a development project, nor would a nationally homogeneous development team be less cultural competent than an inhomogeneous one.

The survey assumed that CQ is relevant for software localisation and internationalisation. While not conclusively dismissing the notion, recent results by Mor *et al.* (2015) found that metacognitive CQ does not predict performance of Westerners assessing Asian values.

5.4.3 Software Localisation and Project Properties

In the literature review, it was hypothesised that software development models might have an influence on localisation (e.g. Abufardeh and Magel, 2010; Fissgus and Seewald-Heg, 2005), and that some development models are less suited to work with external processes, specifically the agile development model (e.g. Turk *et al.*, 2002). In the survey, projects without development methodology exhibited significantly lower localisation effort. This suggests that there is indeed a difference between those projects claiming usage of a development model and those that do not, contradicting the notion that

adopting a development model is pretence without actual meaning (see e.g. Truex *et al.*, 2000).

In any way, this difference might merely mean that projects that do not warrant use of formalised development methodologies, maybe because of project simplicity or a very small development team, also do not warrant high localisation effort. Alternatively, it might indicate that a methodology simplifies application of localisation efforts, or that developers who develop software according to a formalised method are more inclined to invest effort into localisation. In any way, localisation effort did not differ between projects using agile and waterfall methodologies.

As suggested in the literature (e.g. Ryan *et al.*, 2009), there was a positive correlation between localisation effort and number of target languages. The more target languages a project has, the higher the localisation effort is. This seems like a foregone conclusion since more target languages require communication with more translators. It is here important to note that none of the items making up the construct LE are obviously affected by the number of target languages. In other words, if one accepts that a higher LE results in a better localisation quality, then more target languages lead to higher localisation quality as well.

An association between localisation effort and software type was observed for video games and system software, both exhibiting higher localisation effort than other software types. For video games, this makes sense as these are often content-heavy applications requiring text and graphics, but also more culture-related items such as jokes and tropes to be adapted to its customers' locales. Results confirms respective statements by Zhou (2011), Callele *et al.* (2008) and Thayer and Kolko (2004).

A rationale explaining why system software would exhibit higher localisation effort is not as straightforward. Arguably, system software might include embedded software for machines in which localisation quality and correctness are critical, e.g. medical apparatuses, prompting an increased localisation effort. Other types of software did not show a different localisation effort score, contrary to what is expected in the literature (e.g. Giammarresi, 2011).

Assumptions that localisation effort might be influenced by type of user (e.g. Liu and Zhang, 2011), the relationship between developers on one side and customers or users on the other side (e.g. DePalma, 2006; Honkela *et al.*, 1997), and the commercial or non-commercial character of a project (e.g. Exton *et al.*, 2010; Wolff, 2006) could not be confirmed in the survey.

Statements about localisation effort need to be considered in the context of its operationalisation in this research. There might be activities that are undertaken in software development projects to improve localisation in terms of cost, quality or duration that did not find consideration in this operationalisation. While noting that the construction and measurement of LE was justified due to the lack of suitable translation and localisation quality measures suitable to questionnaires, it must be noted that the construct LE itself is speculative since no validity or reliability tests were concluded prior to the survey. A large part of the literature review in chapter 2 dealt with localisation tools and standards. It is noteworthy that despite a great effort of improving localisation tools, the survey data shows only light adoption. While a moderate adoption of machine translation and specialised translation systems, e.g. TM, was observed, the proliferation of specialised localisation file formats, specifically XLIFF, is still lacking.

5.4.4 Generalisability of the Sample

As mentioned in section 3.4, the sample is a convenience sample, which simplifies recruitment and can increase participant numbers. However, convenience samples are non-probabilistic and as such not representative of the entire population of developers and localisers. Identified biases in the sample are for example nationalities (see Table 5-1). There is a clear skew towards German, British and US-American nationalities, probably due to the media in which the call for participation was published. Nationalities of global localisation professionals are expected to follow a different distribution, in particular regarding German being less emphasized.

Another potential bias might be a relationship between survey participation/completion and attitude towards localisation for developers, i.e. that a more positive attitude increases likelihood of survey participation, which might skew the comparison of developer and localiser attitudes. However, since the results show a difference in attitude

already, presence of an attitude-participation bias would mean that the effect were even stronger without it.

Some survey items might have been defeated by socially or politically preferred answers. For example, the test for differences in localisation scope assessment only asks for knowledge. One could repeat these tests through an instrument where the actual intention of the question is not obvious.

Almost all incomplete surveys stopped after the biographical data, i.e. in the second section, where the perceptions and opinions are queried. It is assumed that most non-completers stopped here because of the monotony of the test and the implied length of the survey. It is assumed that there is no content-related reason for non-completion, e.g. non-completers did not mind having their cultural competence measured. It is not perceivable why respondents would be sensitive about their cultural competence, particularly because the survey is anonymous.

Some data points were excluded from analysis (see Table 5-11). With one exception where the role could not be attributed to either the developer or localiser group, all data exclusions were a consequence of the respondents' presumable lack of knowledge regarding the question asked. With one exception, excluded data points were always six or less and are numerically unlikely to have an influence on the result. Because 27 respondents did not specify their projects' software development models when testing for correlation between localisation effort and development method, it is conceivable that the missing data had an influence on the result. However, no systematic skew is apparent to explain why development model and not knowing it might be related.

5.5 Summary

This chapter presented and discussed the results of the quantitative part of this research. It was hypothesized that developers and localisers differ measurably across a number of items relevant to software localisation. However, a majority of differences could not be confirmed. Most importantly, there seems to be no difference regarding the interpretation of success and quality criteria for software. On the other hand, gradients between developers and localisers in cultural competence and attitude towards localisation were confirmed. It was also shown that cultural competence can be, and is,

trained, but does not affect attitude towards localisation. Further quantitative results showed that some software types, the use of a development model, and the number of languages are correlated to the localisation effort expended. On the other hand, the relationship to the user and a commercial or non-commercial character of the software did not.

At the end of this chapter, the results were compared to the expectations in localisation literature. In the next chapter, the qualitative and quantitative findings will be discussed in conjunction while addressing the original research problem: What makes software localisation difficult?

Chapter 6 Conclusions

In the introduction, the proposition was made to examine what makes software localisation difficult. This problem was developed into four research questions aimed to examine the human factor and the project factor in software localisation. The research questions were answered by conducting and analysing interviews using the GT methodology, and by statistical analysis of survey data on cultural competence, attitude and self-efficacy towards localisation of participants, and properties and localisation effort of the projects they worked on. This concluding chapter will reiterate the findings and discuss contribution to knowledge, implications for practice, and limitations of this research, as well as future work.

6.1 Summary of Findings

RQ 1 asked how localisation is conducted individually and collaboratively by developers and localisers, and how this shapes each discipline's activities. To answer the question, a grounded theory of interdisciplinary collaboration in software localisation was generated in which the work of developers and localisers is shaped in part by factors out of their control, and by expected and experienced conflicts during software localisation.

RQ 2 asked how issues are caused during localisation and internationalisation. This research question was also answered by the generated grounded theory of interdisciplinary collaboration: localisation issues can be caused when, in the hierarchical relationship of developers and localisers, each discipline follows its interest rather than a localisation goal.

RQ 3 asked in what regards developers and localisers are distinct. Developers and localisers score differently on cultural competence, but exhibit no difference in self-efficacy in localisation, responsibility towards localisation, and assessment of localisation scope. Developers' attitude towards localisation is lower than that of localisers, and they prioritise maintainability higher.

RQ 4 asked what dependencies exist between localisation effort and properties of development projects. More localisation effort is expended on video games and system software than on other software types, as well as on projects following a development model compared to projects without one. Further, the more target locales a project has,

the more localisation effort is spent on it. The commercial nature of a project and the relationship between user, developer and customer do not affect localisation effort.

6.1.1 Conjunction of Qualitative and Quantitative Results

This research used both qualitative and quantitative methods to answer various research questions. Although the two paradigms were not intended to be combined, some of their results nonetheless complement and contrast with each other.

In the interviews, a profound lack of appreciation for non-engineering aspects of software development was observed and linked to conclusions in the literature. The lower attitude towards localisation of developers found in the survey is in agreement with such an engineering mind-set and a lack of appreciation for non-engineering aspects of software development.

There was a difference between localisation scope reported by interviewers and localisation scope found in the survey. All localisation projects the interviewees had experienced were limited to text translation and proper presentation of text and textual data such as number formatting. No interview participant had ever experienced a localisation project going further, e.g. by adapting colours or icons. Only one participant reported the adaptation of functionality, but the context suggests that their objectives were as much unifying previously separate IT systems as they were catering for different cultures. The survey, on the other hand, reported that more than a quarter of all projects have layout, navigation and functionality adapted, and about a fifth of all projects had icons, images, sounds, and feature sets adapted.

There are three plausible explanations for this disparity: It might be that interview and survey samples differ by chance. Alternatively, the reporting in the interview might be correct but some survey respondents might have been reporting their opinions and assumptions rather than their experiences. Thirdly, the survey question might have been too unspecific. Indeed, in the interviews, a number of participants had reported that text in image files was translated while keeping the pictorial content untouched. This is then translation, not adaptation of graphical content. Unfortunately, the respective survey question Q.63 did not make that distinction.

6.2 Contribution to Knowledge

As discussed in the literature review, there have been few studies of empirical examination of localisation practice and localisation issues. Abufardeh (2008) and O'Sullivan (2001b) examined the impact of localisation on software quality in terms of bugs introduced into the software. O'Sullivan (2001b) and Moorkens (2012a, 2012b) included a description of localisation practice from interviews in their research. Immonen and Sajaniemi (2003a, 2003b) conducted a descriptive examination of localisation practice, but limited interviews to management professionals at Finnish companies.

This thesis went beyond their research by applying well-defined research methodologies for qualitative data collection and analysis, and gathering data from various contributors to international software. The research findings provide a perspective on how localisation is handled in practice based on the perception of its stakeholders, and how this practice affects its product. The evidence of this research consists of self-reported perceptions from stakeholders through qualitative and quantitative means. A grounded theory of interdisciplinary collaboration in software localisation emerged which explains how development and localisation professionals apply strategies of interdisciplinary collaboration based on existing external influences and interdisciplinary conflicts. This theory contributes to answering what makes software localisation difficult.

6.2.1 A Grounded Theory of Interdisciplinary Collaboration

The interviews led to a grounded theory of interdisciplinary collaboration in software localisation in which external influences, strategies of professionals and interdisciplinary conflicts interact with each other. The main concern of participants is the facilitation of interdisciplinary collaboration across development and localisation. Strategies of developers and localisers to that end are shaped by external influences outside of their control, e.g. success criteria and the workings of development and CAT tools. Those influences interact with strategies, and lead to conflicts in interdisciplinary collaboration. Localisation issues are caused when its cost, quality and schedule goals are compromised through and replaced by developers' and localisers' individual interests.

6.2.2 Localisation is Difficult Due to its Multidisciplinary Character

Software localisation is difficult. Previous research has mostly seen this difficulty in the context of the brobdingnagian number of cultural idiosyncrasies, how to represent and manage them in software, and how to provide translations quickly and efficiently.

This research has examined the difficulty in software localisation in the context of two disciplines with different practices, epistemologies and work focuses working together. It becomes apparent when observing software localisation in practice. For developers, software localisation is a product-centric process to be integrated into existing software development processes. It involves the creation of a technical infrastructure and the separation of code and content. For localisers, software localisation is a culture-centric outcome determined by the product it lives in. It involves the management of translation and text through the integration and processing of translation-relevant information. Thus, different views collide in practice, and when there is no even playing field to negotiate processes and aims, the power of one discipline over the other shapes an entire aspect of an undertaking that should involve both.

6.2.3 Localisation Issues are Caused by the Separation of Disciplines

Localisation errors can be caused by the strict separation between localisation and development. In the course of the interviews, it has been shown that the development side has been virtually always the centre of localisation management: by instruction when processes or interfaces were defined towards the localisation side, or by omission when localisation and its management was simply outsourced to an external agency.

6.2.4 Cross-Disciplinary Knowledge Trumps Cultural Competence

Barbour and Yeo (1997) suggested that tackling *ethnocentrism*, i.e. a preference of Western culture, is an important step towards developing truly international software. In this research, that stance found recognition by examining cultural competence. However, taking into account the interview results, the hierarchical position of software engineering above other matters in software development as described by e.g. Low *et al.* (1996) seems a better explanation for localisation issues.

6.2.5 Support for the Notion of a Software Engineering Mind-Set

Although the survey results are not a comprehensive confirmation of the notion that developers have an attitude problem towards other disciplines, to the best of my knowledge, this is the first explanatory empirical support for the notion.

6.3 Implications for Practice

While this research did not aim to create and evaluate recommendations and guidelines, the results provide an understanding of the process of software localisation as source of cost, quality and schedule issues. The findings of this research have significance for organisations developing and localising international software, individual developers and localisers, and those who manage them. Sharp *et al.* (2005) argue that there is a benefit in increasing accountability by making software development processes visible and generally available. A description of how localisation and software development conflict and how localisation issues are caused is essential for improving software localisation overall and increasing its efficiency and effectiveness towards currently underemphasized factors such as quality.

6.3.1 No Complete and Comprehensive List of Cultural Differences

During the research, a common, almost universal, desire of software developers was what they must perceive as the holy grail of internationalisation: a comprehensive and complete list of all aspects that must be variable in an application so that it is completely internationalised. It mirrors the description of Green (1994) that developers expect a self-contained deliverable which is easy to understand, apply, and integrate into their processes. To paraphrase Cooper (2004), developers ask for a universal guide for a problem that needs to be solved on a case-by-case basis. The notion of a list of cultural difference might also lie at the heart of cultural resource banks or repositories discussed in the literature (Ryan *et al.*, 2009; Smith and Dunckley, 2007; Mahemoff and Johnston, 1998). Maybe this view is one of the main causes of the phenomenon noted by Austermühl and Mirwald (2010), that translators' self-image as intercultural communication experts is not shared by the localisation industry, who instead see them as a source of linguistic and technological skill.

Of course, such a list does not exist, and cannot exist considering the variation of culture and the infinite number of potential cultural differences. These are context-dependant, i.e. depend on what the software is supposed to achieve. In fact, as Ito and Nakakoji (1996) write, the term *cultural difference* is insofar misleading. Improvement might instead be obtained by perceiving differences as manifestations of different social backgrounds. Ito and Nakakoji hence do not give a list of cultural differences or elements in software. Instead, they identify the cultural impact on different stages of the human-computer interaction process. This might be helpful when conducting locale-usability tests as discussed by del Galdo (1996).

Boehm (2011) writes that postmodern software development de-emphasising the positivistic notion of certainty has not quite arrived yet. One might argue that a request for a complete and comprehensive list of cultural differences is a further manifestation of a modernist paradigm in software development as discussed by Robinson *et al.* (1998) in two ways: First, such a list is supposed to give certainty where, according to the postmodernist view on software localisation by Barbour and Yeo (1997) and confirmed by the research results of Sun (2004a), no certainty exists because the interpretation of UI in cultural context has to be left to the user, not software developer or even localiser. Second, it illustrates the hierarchical supremacy of software engineering within software development as it marginalises cultural expertise to something that can be separated cleanly, its implementation then left to software engineers.

6.3.2 Localisation as Process Rather than Deliverable

Instead of seeing localisation as a deliverable, localisation should instead be an activity exercised during product design, requirements engineering and implementation (Giammarresi, 2011). Many accounts suggest that, especially in projects where it is outsourced, localisation is reduced to an activity to separate text from code and use available localisation APIs. This might be traced back to the basic idea of internationalisation and the assumption that a separation of locale-dependent and locale-independent parts of software would also extent itself to processes and social relationships. This research's results suggest that O'Sullivan's (2001b) argument about the choice between process model and architecture model is incomplete. He suggests the architecture model, ironically in order to reduce the number of localisation-caused

software bugs. But this choice does not make a process model obsolete. Collaboration, the social dependency, is always conducted in some way, even if the choice is not to collaborate. The communication researcher Paul Watzlawick has been famously ascribed the quote that “one cannot not communicate”, meaning that even the total refusal to interact is an act of communication. Likewise, if there is a technical dependency, one cannot not collaborate insofar as even the most minimal imaginable contribution from one discipline still affects the work activities and output of the other.

6.3.3 Counteracting Control, Agency and Dominance in Localisation

If, as was suggested, internationalisation, existing localisation and processes and tools might follow an attempt to control the workforce in the sense of scientific management, then it might be worth considering alternatives, e.g. empowering the workforce by giving them access to training, budget, information and most of all authority to make decisions for which they are then responsible (Tubbs and Moss, 2003).

When the ultimate decision for the shape of the product lies with the development side, it also needs to be tasked with resolving and avoiding translation and localisation errors. This is, in fact, a key recommendation of Dr. International (2003, p.15), yet rarely seems to be applied in practice. In fact, sharing of responsibility and performance measures have been identified as strategies to counter selfish behaviour by the agent. Further, an increased inclusion of the development side in localisation might include an introspective examination of the motivations for localisation and its respective acknowledgement. There might be a variety of reasons, from concern to the international user to fulfilment of a legal requirement of a target market. Hence, the motivation to localise software should inform not only the budget, but also the overall localisation strategy as well as the level of commitment applied to localisation from the development side.

6.3.4 Creating Cross-Disciplinary Knowledge

In the light of the dominance that the development side exerts over software localisation, and the catering for it by the agency model, it is all the more surprising that in this research, comparatively little insight on the development side was observed that localisation errors might be systemic, i.e. that certain localisation issues might be an inherent property of development and localisation processes. This is unfortunate because

the survey results in particular can be interpreted to mean that developers and the development process have an influence on localisation effort: it correlates with software type and number of languages, which are aspects of development. This suggests that localisation issues, and in particular quality problems, are not results of random events and can be avoided.

Cross-disciplinary awareness, i.e. an understanding of each other's competence and ability, seems to be a central aspect of interdisciplinary collaboration. Developers do not need to be localisers and *vice versa*, but key knowledge can be identified, such as the importance of context or placeholder syntax. The idea behind this is to break up the entrenchment where either side does not know what the other side needs or does.

6.4 Limitations

All samples and sample origins were reported along with all data exclusions, all manipulations and all measures, which were entirely made in good faith towards a scientific result. A number of factors reduce generalisability and representativeness:

The GT method emphasises contextual fit and theory progression over representativeness and generalisability to begin with. Both are further reduced due to the extremely wide range of development, localisation processes and configurations, making a representative sample of development-localisation configurations difficult.

GT is sometimes described as a method to help finding *the* problem. In this thesis, what was found was *a* problem. The categories and concepts identified in the theory reflect the interview accounts and were obtained mostly through line-by-line and paragraph coding, classification and conceptualisation. Coding was conducted with post-formed codes developed during analysis. No existing theoretical framework was used and to the best of my knowledge, there were no existing applicable studies. Microanalysis was conducted during open coding and whenever accounts or details were unclear later.

Although the GT process result was initially descriptive, during a later conceptual analysis, categories, relationships, and to a degree also properties emerged leading to a narrow and substantive theory beyond mere description. However, the identified theory is not the only possible way to organise and interpret the data.

Validation occurred during data collection by comparing new data with existing data. To improve validity further, a number of recommendations by Runeson and Höst (2009) were followed, i.e. the interview protocol was supervisor-reviewed, and cases were thoroughly examined, including looking for contradictions to the developing theory. Inter-rater reliability measures ensuring similar coding were not applied as I was the only coder for this thesis.

Pure translators without a management role are probably underrepresented. Every effort was made to increase participation of translators. Since interviewees often chose to be interviewed during working hours, it can be speculated that translation freelancers are less motivated to participate because participation comes out of their own time, whereas many participating developers and LSP project managers are employed and participation during work time are seen as no personal sacrifice.

The survey used a convenience sample, which is not representative of the entire population of localisation professionals. However, convenience samples allow easier recruitment, leading to more participants. Examples for the limitations of the generalisability of the survey results are a clear nationality skew towards German, British and US-American.

The survey is limited regarding validity and reliability of some of its instruments, i.e. whether measurements are repeatable and actually measure the intended constructs. The CQS is a validated instrument and was used without changes. The instruments measuring ATL, SEL and SEU were adapted from validated instruments. As a construct, LE was not validated. During analysis, it was noted that some of the LE questions, specifically Q.64 to Q.67, need to be rephrased for a better inclusion of translators and localisers. Respectively, the instruments measuring ATL, SEL and SEU could be re-validated and LE improved and validated to increase the robustness of results of a repeat survey, preferably with a probabilistic sample for improved generalisability.

The survey was designed to be applicable and comprehensible for both developers and localisers. A piloting phase showing no comprehensibility issues for translators. However, during the actual data collection, repeated feedback was received from localisers who felt that the survey was not relevant to them.

GT is, by definition, a method of exploratory research. Statistical survey analysis, on the other hand, is by definition a method of explanatory research. However, due to the limitations of the survey, the ad-hoc character of the tested hypotheses, and the liberal use of regression and correlation measurements, the survey results should be considered exploratory following recommendations by Kitchenham *et al.* (2008).

In order to address the limitations of this research, it is suggested to validate the items in the survey and verify the GT results. For the latter, in addition to the methods already employed in this research, Hoda (2011) verified data by presenting research results to audiences at conferences and industry meetings and gathering feedback, and triangulating interview data with observations. Similarly, Martin (2009) informally verified through feedback and triangulated with observations and archival data. In order to avoid further lengthy qualitative data collection, for this research it is suggested to obtain feedback on the results from participants in a quantitative survey that is constructed so as to avoid confirmation bias. This can be done with relatively little effort and later be expanded into a study with a probabilistic sample of practitioners, including validated constructs of the original survey in this research.

6.5 Future Work

Parts of the results of this study suggest a hierarchical relationship between development and localisation. The survey further tested for cultural competence differences of developers and localisers, but the GT results suggest that developers' knowledge of how localisers work is more important. Both hierarchical relationship and cross-discipline knowledge can be tested, e.g. by measuring the locus of control as perceived by localisers, which has been identified as a relevant construct in this context of socio-technical aspects of work, or surveying developers' knowledge of translation and localisation processes.

A more ambitious project would be to develop a practical measure of localisation quality, the lack of which currently appears as one of the biggest limitations both in localisation and in conducting research about it. In this research, it was tried to partly address this by measuring localisation effort instead. However, localisation quality assessments as suggested by Nielsen (1996a, 1996b) and Sun (2004a), i.e. task performance tests across different locales with respective users, are not practicable in surveys. Other suitable

definitions of localisation quality for surveys are not known (Lewis *et al.*, 2009). If an easily applied measure of localisation quality could be developed, it would open up the possibility to reliably test the effects of various tools and processes, and it would become possible to further examine the effects of outsourcing or CAT tools on quality. In this research, since localisation quality was replaced by localisation effort, this was not possible as the independent variable, i.e. use of CAT tools and outsourcing, was part of the dependent construct LE.

Another more ambitious project might be to examine a software artefact and archival data from its development process in order to identify localisation issues and their causes. This could include focus group studies with the professionals involved during original development. Such research might be used to verify the GT results of this research by triangulation, but would be connected to serious effort and would have to overcome a number of practical obstacles such as obtaining access.

The phenomenon of interdisciplinary collaboration should be examined further in a more general sense. Existing research seems to focus on the narrow areas of collaboration in a team and in the academic sector. However, I wonder if there is not a larger riddle here. Some of the findings seem to suggest that the distinction between developer and localiser might be more than just different work priorities and more or less conscious profit maximisation strategies. Deep knowledge of a discipline or subject area might inform our behaviour and cognition in a much more profound way. Questions to ask are what effect affiliations to a discipline have on human perception, communication and behaviour, and whether these are conscious or subconscious phenomena. It would be most interesting to know whether disciplinary knowledge works as a communication filter or influences perceptions of professionals.

Appendix A Survey

Note: Item numbers were not shown on the web form, but are shown here to simplify identification in the text.

Software Localization Survey

In this survey, we examine the interoperation of software development and localization processes and its influence on quality and development effort. If you have ever participated in the creation of software for international markets, including websites, then this survey is for you, regardless of how much you know about software localization.

The survey serves scientific purposes only. The collected data from all participants will be pooled and participants will remain anonymous. Any data will be treated confidentially.

This survey is divided into five parts. Completing it takes approximately 15 to 20 minutes.

For the purposes of this survey, "localization" refers to adapting a product for different markets, languages and cultures (e.g. translation of user interface text). A "localizer" would be the person doing this. Usually, this is also a translator. "Localization" in the context of this survey also includes "internationalization".

Instructions:

- Please follow the instructions and read each question carefully before answering.
- Answer questions speedily and in the given order. Don't skip any questions.
- Don't worry if you are not sure about the precise meaning of a question or statement. Make a guess and follow your instincts.
- If none of the answer options fit perfectly, choose the one that comes closest.
- There are no "right" or "wrong" answers.
- It is ok to choose the "I don't know" option often. A substantial goal of the survey is to determine how much knowledge about software localization is available in the development community.

If you have any questions or comments, please feel free to contact us at locdevresearch@uwl.ac.uk.

Part 1

1. What is your age in years?

   _____

2. What is your gender?

   Male / Female

3. Have you been working on software projects which were localized?

   Yes / No / I don't know

4. What is your nationality?

   _____

5. What is your highest level of education you have completed?

   - *High School, Grammar school, Gymnasium or equivalent*
   - *Bachelor's degree or equivalent (e.g. BA, BSc)*
   - *Master's degree or equivalent (e.g. MSc, MA)*
   - *Doctoral degree or equivalent (e.g. PhD, Dr)*
   - *Other, please specify: _____*

Part 2

Thank you for participation so far. The second section considers perceptions of and opinions about software localization. Please rate the following statements on a scale from "strongly disagree" to "strongly agree".

6. *For me, software localization is not a regular concern on a day-to-day basis.*

7. *Applying knowledge in software localization can help me to create more effective software for international users.*

8. *I am confident about my ability to do well in a software project which is localized.*

9. *If the software project I am working on is localized, this will only mean more work for me.*

10. *I do not think that software localization will be useful for the software projects I am working on.*

11. *I feel at ease learning about software localization.*

12. *Software localization helps increasing the user base of software projects I am working on.*

13. *I am not the type to do well with software localization.*

14. *Software localization will increase the usability of the software I am working on for international users.*

15. *Any gain through software localization can be achieved just as well some other way.*

16. *The thought of software localization being a part of the software project I am working on frightens me.*

17. *Software localization is confusing to me.*

18. *Software localization is necessary for software projects to adhere to local laws and customs.*

19. *I do not feel threatened by the impact of software localization on my software projects.*

20. *I am anxious about software localization in my software projects because it might interfere with my efforts or ideas.*

21. *Software localization helps avoid misunderstandings and offenses for international users of the software I am working on.*

22. *I don't see how software localization can improve my software project for international users.*

23. *I feel comfortable about my ability to handle software localization in my software projects.*

24. *Knowing about software localization will not be helpful in my future work.*

25. *I feel confident employing localization-related functionalities of UI frameworks (e.g. WPF, Cocoa).*

26. *I feel confident documenting context information about the software I'm developing for a translator.*

27. *I feel confident deciding what elements of an application need to be localized.*

28. *I feel confident handling translations from third parties for inclusion in the software I'm developing.*

29. *I feel confident working on projects which use Unicode.*

30. *I feel confident creating a clear user interface layout for software.*

31. *I feel confident conducting usability tests.*

32. *I feel confident phrasing error messages for software in a helpful way.*

33. *I feel confident analyzing feedback from usability tests.*

34. *I feel confident implementing changes suggested by usability experts in the software I'm developing.*

Part 3

Following are 20 statements about your general interactions with other cultures in everyday situations. Read each statement and select the response that best describes your capabilities. Select the answer that BEST describes you AS YOU REALLY ARE (1 = strongly disagree; 7 = strongly agree).

35. *I am conscious of the cultural knowledge I use when interacting with people with different cultural backgrounds.*

36. *I adjust my cultural knowledge as I interact with people from a culture that is unfamiliar to me.*

37. *I am conscious of the cultural knowledge I apply to cross-cultural interactions.*

38. *I check the accuracy of my cultural knowledge as I interact with people from different cultures.*

39. *I know the legal and economic systems of other cultures.*

40. *I know the rules (e.g., vocabulary, grammar) of other languages.*

41. *I know the cultural values and religious beliefs of other cultures.*

42. *I know the marriage systems of other cultures.*

43. *I know the arts and crafts of other cultures.*

44. *I know the rules for expressing nonverbal behaviors in other cultures.*

45. *I enjoy interacting with people from different cultures.*

46. *I am confident that I can socialize with locals in a culture that is unfamiliar to me.*

47. *I am sure I can deal with the stresses of adjusting to a culture that is new to me.*

48. *I enjoy living in cultures that are unfamiliar to me.*

49. *I am confident that I can get accustomed to the shopping conditions in a different culture.*

50. *I change my verbal behavior (e.g. accent, tone) when a cross-cultural interaction requires it.*

51. *I use pause and silence differently to suit different cross-cultural situations.*

52. *I vary the rate of my speaking when a cross-cultural situation requires it.*

53. *I change my nonverbal behavior when a cross-cultural situation requires it.*

54. *I alter my facial expressions when a cross-cultural interaction requires it.*

Part 4

In this part, we'd like to learn more about your experiences. Please answer the questions in this section for your most recent finished software project which was localized.
If none of your previous projects were localized, but your current project is, please answer the questions for your current project.

55. What kind of software was your most recent localized project? Check all that apply.

    - o *Application software*
    - o *Videogame*
    - o *Website*
    - o *Mobile App*
    - o *System Software*
    - o *Firmware*
    - o *Other, please specify:* _____

56. Who are typical users of your most recent localized project?

    - o *Private end-users*
    - o *Other software developers*
    - o *Scientists*
    - o *Companies*
    - o *Government institutions*
    - o *Educational institutions*
    - o *Other, please specify:* _____

57. Are the users of your most recent localized project your customers?

    - o *Yes*
    - o *No*
    - o *Partly*
    - o *I don't know.*
    - o *This question doesn't apply.*

    (Help: For example, customers and users are not identical if:
    - a company commissions a website for promotion.
    - a hardware developer buys software for bundling with its hardware products.)

58. In a general sense, was your most recent localized project commercial or non-commercial?

- *Commercial*
- *Non-commercial*
- *I don't know.*

59. For how many target languages was your most recent localized product localized? Select one:

    - *1 – 5*
    - *6 – 15*
    - *16 – 30*
    - *More than 30*
    - *I have absolutely no idea.*

    (Help: If you don't know the exact number, please make an approximation.)

60. What software development model was followed mostly during your most recent localized project? Select one:

    - *Waterfall model*
    - *Spiral model*
    - *Agile model*
    - *No particular model*
    - *I don't know.*
    - *Other, please specify: _____*

61. Which of the following statements applies for your most recent localized project?

    - *The localization requirements were clearly defined.*
    - *Best practice guidelines for localization were provided.*
    - *A glossary or corporate dictionary was used in the creation of UI dialogues and text.*
    - *Translations were stored for re-use in other projects.*
    - *A dedicated person or team handled the technical aspects of localization.*
    - *All developers could build, compile and run any language version of the software.*
    - *All language versions, including the original language, were released at the same time.*

62. How often was the importance of localization quality for your most recent localized project emphasized by the project leaders?

    - *Never*
    - *Once or twice*
    - *A few times*
    - *Often*

63. Which parts of the software were localized in your most recent localized project?

    - *User interface text*
    - *Formatting, e.g. time and date and sort orders*
    - *Units, e.g. measurements, currency and paper sizes.*
    - *Colors, graphics and sound*
    - *Navigation and layout*
    - *Functionality*
    - *Feature sets*
    - *I don't know.*

64. Who did the translation work for your most recent localized project? Check all that apply:

    - *Machine translation*
    - *Employees with a different primary task, e.g. marketing or documentation*
    - *Users, for no charge ("crowdsourcing")*
    - *Customers who commissioned the project or language(s)*
    - *Freelancers or external translators*
    - *Agencies or localization/translation providers*
    - *Full-time employees with the primary task of translating*
    - *I don't know*

65. How could you communicate with the localizer during your most recent localized project?

    - *There was no way for us to communicate with each other.*
    - *Communication was unidirectional, e.g. I could contact the localizer, but not the other way around.*
    - *Communication was relayed through a third party, e.g. project managers or agencies.*
    - *We could communicate directly by email.*
    - *We could communicate directly by phone.*
    - *We could meet in person on reasonably short notice.*
    - *Localizers were routinely present in meetings and/or part of the development team.*
    - *I don't know.*

66. How did the translators receive the text to be translated during your most recent localized project? Check all that apply:

    - *We didn't have text to translate.*
    - *Translators edited the text directly in the program files.*
    - *We mailed text in files, but there are no standard file formats to use.*
    - *We mailed text in self-developed proprietary formats.*
    - *We mailed text in txt files or XML files.*

- *We mailed text in standard office formats, e.g. word, excel or rich text format.*
- *We mailed text in XLIFF format.*
- *We exchange text through online databases, content management systems or translation memories.*
- *I don't know.*
- *Other, please specify: _____*

67. Context information helps translators in creating appropriate translations. What kind of information sources were available to them in your most recent localized project? Check all that apply.

    - *Direct access to project members (e.g. by phone or mail)*
    - *Resource files (e.g. containing UI elements or text)*
    - *Complete source code*
    - *A written description of the software and its functionality*
    - *Internal design documents*
    - *Screenshots*
    - *A working version (e.g. a prototype, pre-release version, or the full product)*
    - *None of the above.*
    - *I don't know.*
    - *Other, please specify: _____*

68. What quality assurance efforts were in place for your most recent localized project? Check all that apply:

    - *Localization bugs were fixed when reported by customers.*
    - *Some language versions were partially tested before release.*
    - *All language versions were partially tested before release.*
    - *Some language versions were fully tested before release.*
    - *All language versions were fully tested before release.*
    - *Automated scripts tested for missing translations, UI fit etc.*
    - *All translations and/or localizations were reviewed by a second translator/localizer.*
    - *None of the above.*
    - *I don't know.*
    - *Other, please specify: _____*

<u>Part 5</u>

You have already finished four of five survey parts. The last part covers a few facts and opinions about your previous experience with software localization.

69. What are your usual roles in software development? Check all that apply.

    - *Software engineer*
    - *User interface designer*
    - *Software architect*

- *Business analyst*
- *Project manager*
- *Translator/Localizer*
- *Technical editor*
- *Other, please specify: _____*

70. For how many years have you been working on software projects for international users?

    _____

71. Did you receive any training about software localization? Check all that apply:

    - *I have read books and/or articles dealing exclusively with software localization.*
    - *I have read books and/or articles dealing in part with software localization.*
    - *I have received informal training, e.g. from colleagues.*
    - *I have received formal training, e.g. a course.*
    - *None of the above.*

72. In your opinion, what parts of software should be localized?

    - *User interface text*
    - *Formatting, e.g. time and date, sort orders*
    - *Units, e.g. measurements, currency, paper sizes, and currency*
    - *Colors, graphics and sound*
    - *Navigation and layout*
    - *Functionality*
    - *Feature sets*
    - *None of the above*
    - *Other, please specify: _____*

73. Sort the following nouns according to what you consider most important for software:

    - *Maintainability*
    - *Reliability*
    - *Correctness*
    - *Execution speed*
    - *Usability*
    - *Power*
    - *Popularity*
    - *Success*

74. Sort the following priorities according to what you consider most important for software development outcomes:

    - *Costs within budget*

- *Quality on target*
- *Release on time*

75. Do you feel that generally, responsibility of software quality for international users is part of your roles?

    Yes / No

Thank you for making it so far. We have two more questions which are stored separately from your previous answers to ensure anonymity. Please follow the link below:

-> Please follow this link for two final questions. <-

Thank you very much for completing the survey. The replies on this page are stored separately and can't be linked to your previous answers.

Please feel free to leave a comment:

- Do you have any feedback about this survey?
- Are there important aspects about software localization that have not been mentioned?
- Would you like to leave your email to obtain a copy of the survey results?
- Is there anything else you would like to let us know?

You can also contact us directly at mailto:locdevresearch@uwl.ac.uk.

Experiences in software development are often specific and complex. For that reason, we would also like to interview software developers in person or by phone to gain a more comprehensive understanding. If you would be willing to discuss your experiences, please fill in your email address below and we will get in touch with you.

(Help: Your email address will be saved separately and won't be linked to your replies in the survey.)

Thank you very much for your participation!

If you have any further inquiries, please feel free to contact us at locdevresearch@uwl.ac.uk.

Appendix B Informed Consent Information Sheet

Information about Informed Consent for (telephone) interviews:

Ethical regulations require us to inform all participants of our research of the following:

In our research, we examine the interplay of software development and software localisation and the interoperation of these two areas. Our research aims to understand the impact of processes, infrastructure and interdisciplinary collaboration on required effort and resulting quality. To this end, we conduct interviews with professionals from related disciplines about their practices and experiences.

Our research is conducted according to the ethics codes of the Faculty of Professional Studies at the University of West London, the Association for Computing Machinery (ACM), and The British Sociological Association. The research has been reviewed and approved by the University of West London Research Degree Committee.

You have the right to withdraw from participation at any time. Participation in this research is voluntary.

You have a right to remain anonymous. All gathered data will be treated confidentially. Publication of any data will be in anonymized form. No individuals and/or companies will be identifiable. Confidential information will not be shared with anybody. Research results will be published. Participants will receive a copy of the research results if they want.

If you have any questions, you may ask them at any time.

Appendix C Interview Excerpt

Author: So, I'm just curious, what you just told me about that you have this e-book xml or the xml file with the text for the buttons, erm, did you that before, or did somebody, I mean, did somebody tell you, did your engineer tell you that when you were [at company headquarters] last time, and asked them about localisation? Or did they, how did you know that, how it works?

Interviewee: Me?

Author: Yeah.

Interviewee: Because, erm, erm, er, I have thorough training about the back office, you know. So... [CODE: source of knowledge of localisation process and infrastructure]

Author: Ah ok. So that was part of the training.

Interviewee: Yeah, it was for the training [CODE: source of knowledge of localisation process and infrastructure] [NOTE: Maybe participant received training because he is involved with customers and customers are involved with localisation.]*, and the, er, er, er, so this is a process. You want any new languages, ok, you provide to us a file, an xml file, with all the translation, and we are going to integrate after then in the back office* [CODE: localisation process]*. Ok? So, we don't have a big program, you know, with a culture between, between countries, because if [customer 1] want to translate in German, is easy for, for them, because there is a subsidiary* [CODE: localisation by customer]*, there is German people there, so the, they speak French and German, so they do the work, and after they give to us the file.*

Author: Ok. Just that I get this right, because for me if, if you, we look at the brochure again, or the catalogue, well, we don't really, yeah, there it is. I mean there are two separate things, right? One is the actual brochure, the catalogue, and obviously [customer 1] has to, has to translate themselves. Of that, you know, the customers, but for the reader, for the buttons up there, you know, those, these are, in a way those are part of your program. Right? So I'm just wondering: Do they do the German translations, or do you get them somewhere else, or do you get them from [customer 1], for the buttons?

Interviewee: Yeah, it's [customer 1] who give, who give the file. To us.

Author: Because I'm wondering, let's assume that for the next version of your frontend you have a new button.

Interviewee: Ok.

Author: Then you need this translated in 32 languages, right?

Interviewee: We need to ask, erm, at each of our customer who want to integrate this new feature. For example tomorrow we are going to launch, I don't know, a new feature, a new button, I don't know, to share on Facebook, maybe. To share the [unintelligible] on Facebook. So, if our customer wants, er, this new feature, we are going to ask to them, so we are going to launch a new product, and so, so we need a translation for, er, for, er, for the display on the front office. So maybe takes time? But you know, it's good.

Author: Yeah, I understand. But then you kind of, that means that you're kind of dependant on your customers to get the translations?

Interviewee: No, because if, if we want, because it's not for us, it's our customer who ask us, I want these languages, I want these languages. So if they don't want the languages, it's not a problem for us. [CODE: defer responsibility to customer]

Author: So you would, just that I get this right, so you would say the, erm, the responsibility of getting the translations lies with the customer?

Interviewee: Yes.

Appendix D Memo example

Excerpt from the interview:

> *Author: Ok. You mentioned that there, basically, localisation is, I can show you that here [refers to previously drawn diagram of the participant's localisation architecture], has these three parts: integration framework...*
>
> *Interviewee: Yes. Ok, so one of these apps is called the [framework name], which is... every [local subsidiary] got a different billing system, right? So if you're gonna write a generic thing that says, you know, pay, authorize refund, then you need to have code at the backend which will integrate to those backend systems. That by definition will be different for every [local subsidiary], and therefore every [local subsidiary] needs to, you know, write that part themselves. So we provide the framework, and they write that code themselves.*

This led to the following memo:

If the internationalisation is very general, then localisation work requires a lot of coding!

It appears that localisation can be perceived as a mathematical problem: infinite possible customisations allow infinite localisations – and then this particular developer considers the task finished. As a consequence, in this project, the interface between development and localisation is in the technical domain, not between the technical and linguistic/cultural domains; as opposed to an interface which is next to the, let's say, technical and HCI domain, which then can't incorporate cultural issues which are not within HCI.

Does this mean that development might need interfaces to every domain affected by localisation?

Appendix E Sample Request for a Call of Participation

This is a sample request to publish a call for participation. Requests were adapted depending on whether the recipient was an individual or an organisation and was situated on the technical or linguistic side. Here, the recipient is a translator interest group, so the request mentions the localisation/linguist aspects of the research first:

> *Dear team at [organisation],*
>
> *My name is Malte Ressin, I am a PhD student at the University of West London. My thesis topic is the interoperation of software development processes and software localisation, i.e. how issues such as quality, effort and cost are affected by the way in which international software is developed and localised, and the way in which those two disciplines software localisation and software engineering collaborate.*
>
> *For my research, I am among others conducting a survey aimed at translators, project managers, software developers/engineers, UI designers etc., who have worked on localised software. For this, I am still looking for participants. Since [organisation], as a globalization and localization community, probably also has many members with relation to software localisation, I would like to ask if there is an unobtrusive way in which I can inform the [organisation] community about my survey? I tried to look into the member area at [organisation's website], hoping to find a forum. However, as I am not a member of an organisation which is part of [organisation], it appears I have no proper access.*
>
> *It would be very helpful if you could help me find participants for my survey. It can be found behind this link:*
>
> *http://samsa.uwl.ac.uk/locdevsurvey/survey.html*
>
> *The survey is completely anonymous and any gathered information will be treated confidently and only be used for my research. The survey takes about 20 minutes to complete. The results will of course be shared with participants, and if [organisation] could help me spread the link, I'd be more than happy to share results with [organisation], too.*
>
> *I will gladly mail you the survey as word file if you would like to take a look at it first. Please contact me if you have any questions.*
>
> *Thanks a lot and best regards,*
>
> *Malte Ressin*

Appendix F Interview, Transcription and Analysis Tools

Note: Skype, VLC media player and Notepad++ were updated regularly during research so that specific version numbers cannot be provided.

Interview Tools

Recording device:	Samsung YP-U3
VoIP software:	Skype, available at www.skype.com
Recording plugin:	MP3 Skype Recorder 2.1.1, available at www.voipcallrecording.com

Transcription Tools

Playback software:	VLC media player, available at www.videolan.org
Coding software:	WeftQDA 1.0.1, available at www.pressure.to/qda
	NVivo 8, available at www.qsrinternational.com/products_nvivo.aspx
Editor:	Notepad++, available at www.notepad-plus-plus.org

Survey Tools

Survey software:	LimeSurvey v1.90, available at www.limesurvey.org

Statistics Tools

Statistics software:	SPSS 22, available at www.ibm.com/software/analytics/spss

Appendix G Publication Sources for Initial Literature Review

Conferences and Workshops

IWIPS	www.iwips.org
Translating and the Computer	www.translatingandthecomputer.com

Journals

Journal of Specialised Translation	www.jostrans.org
Localisation Focus	www.localisation.ie/oldwebsite/resources/locfocus
Localisation Ireland	www.localisation.ie/oldwebsite/resources/locfocus
Translation Journal	translationjournal.net/journal

Websites

The Localisation Research Centre	www.localisation.ie
The Machine Translation Archive	www.mt-archive.info

Publications

Ressin, M. (2012). *Empirically Researching Development of International Software*. In: Proceedings of the 34th International Conference on Software Engineering (ICSE), 2 June 2012, Zürich, Switzerland.

Ressin, M. and Abdelnour-Nocera, J. (2011). *Comparing Development Roles in Software Localisation*. In: 2nd International Workshop on Comparative Informatics, 2011, Denmark, Copenhagen.

Ressin, M., Abdelnour-Nocera, J. and Smith, A. (2010). *Localization and Agile Development*. In: Designing for Global Markets 9: Proceedings of the Ninth International Workshop on Internationalisation of Products and Systems, Designing for Global Markets, 2010, London: Product & Systems International, p.173 – 176.

Ressin, M., Abdelnour-Nocera, J. and Smith, A. (2011). *A Taxonomy of Software Localization Issues with Connections to the Development Process*. In: Designing for Global Markets 10: Proceedings of the Tenth International Workshop on Internationalisation of Products and Systems, 2011, Kuching, Malaysia.

Ressin, M., Abdelnour-Nocera, J. and Smith, A. (2011). *Defects and Agility: Localization Issues in Agile Development Projects*. In: Agile Processes in Software Engineering and Extreme Programming, 12th International Conference, XP 2011, Lecture Notes in Business Information Processing 77, Heidelberg: Springer, p.316 – 317.

Ressin, M., Abdelnour-Nocera, J. and Smith, A. (2011). *Lost in Agility? Approaching Software Localization in Agile Software Development*. In: Agile Processes in Software Engineering and Extreme Programming, 12th International Conference, XP 2011, Lecture Notes in Business Information Processing 77, Heidelberg: Springer, p.320 – 321.

Ressin, M., Abdelnour-Nocera, J. and Smith, A. (2011). *Of code and context: collaboration between developers and translators*. In: CHASE ’11 Proceedings of the 4th international workshop on Cooperative and human aspects of software engineering, 2011, Waikiki, Honolulu, Hawaii, USA: ACM.

References

Abdelnour-Nocera, J., Dunckley, L. and Sharp, H. (2007). An Approach to the Evaluation of Usefulness as a Social Construct Using Technological Frames. *International Journal of Human-Computer Interaction*, 22 (1 & 2), p.153–172.

Abdelnour-Nocera, J. L. (2007). *The Social Construction of Usefulness*. Saarbrücken, Germany: VDM Verlag.

Abdelnour-Nocera, J. L., Hall, P. A. V. and Dunckley, L. (2003). Making Sense in Intercultural ERP Implementation. In: Evers, V., Röse, K., Honold, P., Coronado, J. and Day, D. L. (eds.), *Designing for Global Markets 5 - Proceedings of the Fifth International Workshop on Internationalisation of Products and Systems*, 2003, Berlin, Germany, p.135–152.

Abdelnour-Nocera, J., Michaelides, M., Austin, A. and Modi, S. (2011). A Cross-national Study of HCI Education Experience and Representation. In: *2nd International Workshop on Comparative Informatics*, 2011, Copenhagen, Denmark.

Abdelnour-Nocera, J. and Sharp, H. (2008). Adopting Agile in a Large Organization. In: *Agile Processes in Software Engineering and Extreme Programming*, 2008, Berlin, Germany: Springer, p.42–52.

Abufardeh, S. and Magel, K. (2008a). Culturalization of Software Architecture: Issues and Challenges. In: *International Conference on Computer Science and Software Engineering*, 2008, p.436–439.

Abufardeh, S. and Magel, K. (2008b). Software localization: the challenging aspects of Arabic to the localization process (Arabization). In: *IASTED Proceeding: Software Engineering SE 2008,* 2008, Innsbruck, Austria, p.275–279.

Abufardeh, S. and Magel, K. (2009). Software Internationalization: Crosscutting Concerns across the Development Lifecycle. In: *2009 International Conference on New Trends in Information and Service Science (NISS 2009)*, 2009, Beijing, China: IEEE, p.447–450.

Abufardeh, S. and Magel, K. (2010). The Impact of Global Software Cultural and Linguistic Aspects on Global Software Development Process (GSD): Issues and Challenges. In: *4th International Conference on New Trends in Information Science and Service Science (NISS) 2010*, 2010, Gyeongju, Korea: IEEE, p.133–138.

Abufardeh, S. O. (2008). *A framework for the integration of internationalization and localization activities into the software development process*. PhD thesis, Fargo, ND, USA: North Dakota State University.

Adair, J. G. (1984). The Hawthorne Effect: A reconsideration of the methodological artifact. *Journal of Applied Psychology*, 69 (2), p.334–345.

Adolph, S., Hall, W. and Kruchten, P. (2011). Using grounded theory to study the experience of software development. *Empirical Software Engineering*, 16 (4), p.487–513.

Agarwal, R. and Karahanna, E. (2000). Time Flies When You're Having Fun: Cognitive Absorption and Beliefs About Information Technology Usage. *MIS Quarterly*, 24 (4), p.665–694.

Ahmed, T., Mouratidis, H. and Preston, D. (2008). Website Design and Localisation: A Comparison of Malaysia and Britain. *International Journal of Cyber Society and Education*, 1 (1), p.3–16.

Albir, A. H. and Alves, F. (2009). Translation as a cognitive activity. In: *The Routledge Companion to Translation Studies*, Revised ed., London, UK: Routledge, p.54–73.

Alchian, A. A. and Demsetz, H. (1972). Production, information costs, and economic organization. *The American Economic Review*, 62 (5), p.777–795.

Ale Ebrahim, N., Ahmed, S. and Taha, Z. (2009). Virtual teams: a literature review. *Australian Journal of Basic and Applied Sciences*, 3 (3), p.2653–2669.

Ali, A. and Kohun, F. (2007). It is Time to Add Kurdish Culture to VS .NET Globalization. *Issues in Informing Science and Information Technology*, 4, p.353–363.

Allen, J. (1999). Adapting the Concept of 'Translation Memory' to 'Authoring Memory' for a Controlled Language Writing Environment. In: *Proceedings of the 21st International Conference on Translating and the Computer*, 1999, London, UK.

Allen, J. D., Anderson, D., Becker, J., Cook, R., Davis, M., Edberg, P., Everson, M., Freytag, A., Iancu, L., Ishida, R., Jenkins, J. H., Lunde, K., McGowan, R., Moore, L., Phillips, A., Pournader, R., Suignard, M. and Whistler, K. (2015). *The Unicode Standard Version 8.0 - Core Specification*. Mountain View, CA, USA: Unicode Consortium. [Online]. Available at: http://www.unicode.org/versions/Unicode8.0.0/UnicodeStandard-8.0.pdf [Accessed: 14 October 2015].

Allen, T. J. (1977). *Managing the flow of technology: Technology transfer and the dissemination of technological information within the research and development organization*. Cambridge, MA, USA: MIT Press.

Amaya, V., Chapman, H., Coady, S., Comerford, T., Estreen, F., David-Moravia, F., Kearney, R., Ryan, K., LIeske, C., Loomis, S., Michael, A., Morado Vásquez, L., O'Carroll, D., Ow, M., Prause, I., Raya, R., Ryoo, J. W., Savourel, Y., Schnabel, B., Schurig, J., Stahlschmidt, U., Waldhör, K., Walters, D. and Wasala, A. (2014). XLIFF Version 2.0. *XLIFF Version 2.0*. [Online]. Available at: http://docs.oasis-open.org/xliff/xliff-core/v2.0/xliff-core-v2.0.html [Accessed: 14 October 2015].

Amichai-Hamburger, Y. (2010). Designing a Net Intergroup Contact Platform: Dealing with cultural Differences and Individual Similarities. In: *1st International Workshop on Comparative Informatics*, 2010, Copenhagen, Denmark.

Anastasiou, D. (2009). Localisation, Centre for Next Generation Localisation and Standards. In: *Explorations across Languages and Corpora - PALC 2009*, 2009, Lodz, Poland: Peter Lang Internationaler Verlag der Wissenschaften, p.401–410.

Anastasiou, D. (2010a). Open and flexible localization metadata. *MultiLingual*, 21 (4), p.50–52.

Anastasiou, D. (2010b). Survey on the Use of XLIFF in Localisation Industry and Academia. In: *Language Resource and Language Technology Standards – state of the art, emerging needs, and future developments Workshop*, 2010, Malta.

Anastasiou, D. and Morado Vázquez, L. (2010). Localisation Standards and Metadata. In: *Metadata and Semantic Research*, Communications in Computer and Information Science 108, Berlin, Germany: Springer, p.255–274.

Anastasiou, D. and Schäler, R. (2009). Lokalisierung - Lokalisierungskonzept, Internationalisierung und Lokalisierung, Software-Lokalisierung [Localisation - Localisation Concept, Iinternationalisation and Localisation, Software Localisation]. *Sprache & Sprachen - Zeitschrift der Gesellschaft für Sprache und Sprachen GeSuS e.V. [Speech and Language - Magazine of the Society for Speech and Language GeSuS e.V.]*, 39, p.45–51.

Anastasiou, D. and Schäler, R. (2010). Translating Vital Information: Localisation, Internationalisation, and Globalisation. *Syn-Thèses*, 3, p.11–25.

Andelfinger, U. (2002). On the Intertwining of Social and Technical Factors in Software Development Projects. In: Dittrich, Y., Floyd, C. and Klischewski, R. (eds.), *Social Thinking-Software Practice*, Cambridge, MA, USA: MIT Press, p.185–203.

Anderson, R. E., Engel, G., Gotterbarn, D., Hertlein, G. C., Hoffman, A., Jawer, B., Johnson, D. G., Lidtke, D. K., Little, J. C., Martin, D., Parker, D. B., Perrolle, J. A. and Rosenburg, R. S. (1992). ACM Code of Ethics and Professional Conduct. *Communications of the ACM*, 33 (5), p.94–99. [Online]. Available at: http://www.acm.org/about/code-of-ethics [Accessed: 14 October 2015].

Ang, S. and Van Dyne, L. (2008). Conceptualization of Cultural Intelligence: Definition, Distinctiveness, and Nomological Network. In: Ang, S. and Van Dyne, L. (eds.), *Handbook of Cultural Intelligence*, M.E. Sharpe, p.3–15.

Ang, S., Van Dyne, L., Koh, C., Ng, K.-Y., Templer, K. J., Tay, C. and Chandrasekar, N. A. (2007). Cultural Intelligence: Its Measurement and Effects on Cultural Judgment and Decision Making, Cultural Adaptation and Task Performance. *Management and Organization Review*, 3 (3), p.335–371.

APA. (2009). *Publication Manual of the American Psychological Association*. 6th ed. Washington, DC, USA: APA Books.

Arthur, K. (1998). Building in the international component. *Localisation Ireland*, 2 (2), p.8. [Online]. Available at: http://www.localisation.ie/oldwebsite/resources/locfocus/issues/1998may.pdf [Accessed: 14 October 2015].

Aryana, B. and Liem, A. (2011). Country-Specific Usability Evaluation: Experience from Iran and Turkey. In: *2nd International Workshop on Comparative Informatics*, 2011, Copenhagen, Denmark.

Austermühl, F. and Mirwald, A. (2010). Images of Translators in Localization Discourse. *T21N - Translation in Transition*. [Online]. Available at: http://www.t21n.com/homepage/articles/T21N-2010-08-Austermuehl,Mirwald.pdf [Accessed: 14 October 2015].

Austin, R. D. (2001). The effects of time pressure on quality in software development: An agency model. *Information Systems Research*, 12 (2), p.195–207.

Badre, A. and Laskowski, S. (2001). *The Cultural Context of Web genres: Content vs. Style*. Georgia Institute of Technology.

Badre, A. N. (2000). *The Effects of Cross Cultural Interface Design Orientation on World Wide Web User Performance*. Georgia Institute of Technology. [Online]. Available at: ftp://130.207.127.23/pub/gvu/tech-reports/2001/01-03.pdf [Accessed: 14 October 2015].

Bandura, A. (1977). Self-efficacy: Toward a unifying theory of behavioral change. *Psychological Review*, 84 (2), p.191–215.

Banker, R. D. and Kemerer, C. F. (1992). Performance evaluation metrics for information systems development: A principal-agent model. *Information Systems Research*, 3 (4), p.379–400.

Barber, W. and Badre, A. N. (1998). Culturability: The Merging of Culture and Usability. In: *Proceedings of the 4th Conference on Human Factors and the Web*, 1998, Basking Ridge, NJ, USA. [Online]. Available at: http://research.microsoft.com/en-us/um/people/marycz/hfweb98/barber/ [Accessed: 14 October 2015].

Barbour, R. H. and Yeo, A. (1997). Strategies of internationalisation and localisation: A postmodernist's perspective. *University of Waikato Working Paper Series*, 17. [Online]. Available at: http://researchcommons.waikato.ac.nz/handle/10289/1115 [Accessed: 14 October 2015].

Barnlund, D. C. (1970). A transactional model of communication. In: *Language behavior: A book of readings*, The Hague, The Netherlands: Mouton, p.43–61.

Baron, R. A. (1995). How environmental variables influence behaviour at work. In: Collett, P. and Furnham, A. (eds.), *Social Psychology at Work: Essays in honour of Michael Argyle*, London, UK: Routledge, p.176–205.

Bauer, S. C. and Rodrigo, E. Y. (2004). Circumstances, challenges and consequences of a quality-geared and technology-aided process of translating: a case study. *Hieronymus - professional quarterly journal of the ASTTI*, 2/2004. [Online]. Available at: http://www.tradulex.com/articles/Cerella-Yuste.pdf [Accessed: 14 October 2015].

Beecham, S., Baddoo, N., Hall, T., Robinson, H. and Sharp, H. (2008). Motivation in Software Engineering: A systematic literature review. *Information and Software Technology*, 50 (9 - 10), p.860–878.

Bijker, W. E. (1997). *Of bicycles, bakelites, and bulbs: Toward a theory of sociotechnical change*. Cambridge, MA, USA: MIT Press.

Bikmatov, R., Glenn, N., Gladkoff, S. and Melby, A. (2013). Visualization of ITS 2.0 Metadata for Localization Process. *Localisation Focus*, 12 (1), p.74–77. [Online]. Available at: http://www.localisation.ie/locfocus/issues/12/1 [Accessed: 14 October 2015].

Blackburn, J. D., Hoedemaker, G. and Van Wassenhove, L. N. (1996). Improving speed and productivity of software development: a survey of European software developers. *IEEE Transactions on Software Engineering*, 22 (12), p.875–885.

Bocij, P., Greasley, A. and Hickie, S. (2008). *Business Information Systems: Technology, Development and Management*. 4th ed. Harlow, UK: Pearson Education.

Boehm, B. (2006). The future of software processes. In: Li, M., Boehm, B. and Osterweil, L. J. (eds.), *Unifying the Software Process Spectrum*, Lecture Notes in Computer Science 3840, Berlin/Heidelberg, Germany: Springer, p.10–24.

Boehm, B. (2011). Some Future Software Engineering Opportunities and Challenges. In: Nanz, S. (ed.), *The Future of Software Engineering*, Berlin, Germany: Springer, p.1–32.

Bohan, N., Breidt, E. and Volk, M. (2000). Evaluating Translation Quality as Input to Product Development. In: *Proceedings of the 2nd International Conference on Language Resources and Evaluation LREC-2000*, 2000, Athens, Greece. [Online]. Available at: http://www.lrec-conf.org/proceedings/lrec2000/html/summary/136.htm [Accessed: 14 October 2015].

Bowker, L. (2005). Productivity vs quality? A pilot study on the impact of translation memory systems. *Localisation Focus*, 4 (1), p.13–20. [Online]. Available at: http://lrc.csisdmz.ul.ie/sites/default/files/publications/Vol4_1Bowker.pdf [Accessed: 14 October 2015].

Braverman, H. (1999). *Labor and Monopoly Capitalism: The Degradation of Work in the Twentieth Century*. 2nd ed. New York, NY, USA: Monthly Review Press.

Brewer, W. F. and Lambert, B. L. (2001). The Theory-Ladenness of Observation and the Theory-Ladenness of the Rest of the Scientific Process. *Philosophy of Science, Supplement: Proceedings of the 2000 Biennial Meeting of the Philosophy of Science Association. Part I: Contributed Papers*, 68 (3), p.176–186.

Briggs Myers, I. (1962). *The Myers-Briggs Type Indicator: Manual*. Palo Alto, CA, USA: Consulting Psychologists Press.

Brooks, F. P. (1995). *The mythical man-month: essays on software engineering*. Anniversary ed. Reading, MA, USA: Addison-Wesley.

Browne, K. (2005). *An Introduction to Sociology*. 3rd ed. Cambridge, UK: Polity Press.

Bryman, A. and Cramer, D. (1995). *Quantitative Data Analysis for Social Scientists*. Revised ed. New York, NY, USA: Routledge.

BSA. (2002). Statement of Ethical Practice for the British Sociological Association. *The British Sociological Association - Giving Sociology a Voice*. [Online]. Available at: http://www.britsoc.co.uk/about/equality/statement-of-ethical-practice.aspx [Accessed: 14 October 2015].

Bumeder, B., Dietz, E. and Sander, M. (2003). Cultural Repository - How can Culture be Integrated into an Engineering Portal. In: Evers, V., Röse, K., Honold, P., Coronado, J. and Day, D. L. (eds.), *Designing for Global Markets 5 - Proceedings of the Fifth International Workshop on Internationalization of Products and Systems*, 2003, Berlin, Germany, p.167–173.

Bunting, R., Coallier, F. and Lewis, G. (2002). Interdisciplinary influences in software engineering practices. In: *Proceedings of the 10th International Workshop on Software Technology and Engineering Practice*, 2002, IEEE, p.62–69.

Caddell, M. and Hall, P. A. V. (2005). New Connections, Old Exclusions? Language, Power and ICTs. In: *Proceedings of the DSA Annual Conference 2005*, 2005. [Online]. Available at: http://www.bhashasanchar.org/pdfs/DSA_confpaper_v4.2.pdf [Accessed: 14 October 2015].

Caesar, M. and Fehrenbach, C. (2005). Management von Lokalisierungsprojekten [Management of localisation projects]. In: Reineke, D. and Schmitz, K.-D. (eds.), *Einführung in die Softwarelokalisierung [Introduction to software localisation]*, Tübingen, Germany: Narr, p.27–38.

Callele, D., Neufeld, E. and Schneider, K. (2008). Emotional Requirements. *IEEE Software*, 25, p.43–45.

Capretz, L. F. (2003). Personality types in software engineering. *International Journal of Human-Computer Studies*, 58 (2), p.207–214.

Carey, J. M. (1998). Creating global software: a conspectus and review. *Interacting with Computers*, 9, p.449–465.

Chavan, A. L., Gorney, D., Prabhu, B. and Arora, S. (2009). The Washing Machine That Ate My Sari - Mistakes in Cross-Cultural Design. *Interactions*, 16 (1), p.26–31.

Chiaro, D. (2009). Issues in audiovisual translation. In: *The Routledge Companion to Translation Studies*, London, UK: Routledge, p.141–165.

Choi, B., Lee, I., Kim, J. and Yunsuk, J. (2005). A Qualitative Cross-National Study of Cultural Influences on Mobile Data Service Design. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2005, Portland, OR, USA, p.661–670.

Choong, Y.-Y. and Salvendy, G. (1998). Design of icons for use by Chinese in mainland China. *Interacting with Computers*, 9 (4), p.417–430.

Christiansen, E. (2010). Comparison as a process of translation. In: *1st International Workshop on Comparative Informatics*, 2010, Copenhagen, Denmark.

Clemmensen, T. (2010). Regional Styles of Human-Computer Interaction. In: *Proceedings of the 3rd International Conference on Intercultural Collaboration*, 2010, ACM, p.219–222.

Clemmensen, T. and Roese, K. (2010). An overview of a decade of journal publications about culture and human-computer interaction (HCI). In: Katre, D., Orngreen, R., Yammiyavar, P. and Clemmensen, T. (eds.), *Human Work Interaction Design: Usability in Social, Cultural and Organizational Contexts*, IFIP Advances in Information and Communication Technology 316, Berlin/Heidelberg, Germany: Springer, p.98–112. [Online]. Available at: http://openarchive.cbs.dk/bitstream/handle/10398/7948/WP_2009_003.pdf?sequence=1 [Accessed: 14 October 2015].

Coleman, G. and O'Connor, R. (2008). Investigating software process in practice: A grounded theory perspective. *The Journal of Systems and Software*, 81 (5), p.772–784.

Collins, R. W. (2001). Software Localization: Issues and Methods. In: *European Conference on Information Systems ECIS 2001 Proceedings*, 4, 2001, Bled, Slovenia, p.36–44.

Collins, R. W. (2002). Software Localization for Internet Software: Issues and Methods. *IEEE Software*, 19 (2), p.74–80.

Combe, K. R. (2011). Relationship management. In: Dunne, K. J. and Dunne, E. S. (eds.), *Translation and Localization Project Management: The Art of the Possible*, American Translators Association Scholarly Monograph Series XVI, Amsterdam, The Netherlands: John Benjamins Pub. Co, p.319–345.

Cook, J. D. (2011). Software exoskeletons. *The Endeavour*. [Online]. Available at: http://www.johndcook.com/blog/2011/07/21/software-exoskeletons/ [Accessed: 14 October 2015].

Cooper, A. (2004). *The Inmates are Running the Asylum - Why High-tech Products Drive us Crazy and How to Restore the Sanity]*. Paperback ed. Indianapolis, IN, USA: Sams Publishing.

Cosby, K. S. and Croskerry, P. (2004). Profiles in Patient Safety: Authority Gradients in Medical Error. *Academic Emergency Medicine*, 11 (12), p.1341–1345.

Crabtree, C. A., Seaman, C. B. and Norcio, A. F. (2009). Exploring language in software process elicitation: A grounded theory approach. In: *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009, Lake Buena Vista, FL, USA: IEEE Computer Society, p.324–335.

Creswell, J. W. and Clark, V. L. P. (2007). *Designing and conducting mixed methods research*. 2nd ed. London, UK: Sage.

Cyr, D. (2008). Modeling Website Design across Cultures: Relationships to Trust, Satisfaction and E-loyaly. *Journal of Management Information Systems*, 24 (4), p.47–72.

Cyr, D. and Trevor-Smith, H. (2004). Localization of Web design: An empirical comparison of German, Japanese, and United States Web site characteristics. *Journal of the American Society for Information Science and Technology*, 55 (13), p.1199–1208.

Dagenais, B., Ossher, H., Bellamy, R. K. E., Robillard, M. P. and de Vries, J. P. (2010). Moving into a new software project landscape. In: *Proceedings of the 32nd International Conference on Software Engineering*, 1, 2010, Cape Town, South Africa: ACM, p.275–284.

Deal, T. E. and Kennedy, A. A. (2000). *Corporate Cultures: The Rites and Rituals of Corporate Life*. New York, NY, USA: Perseus Publishing.

De Dreu, C. K. and Weingart, L. R. (2003). Task Versus Relationship Conflict, Team Performance, and Team Member Satisfaction: a Meta-Analysis. *Journal of Applied Psychology*, 88 (4), p.741–749.

Dennis, H. S. (1975). The construction of a managerial communication climate inventory for use in complex organizations. In: *Annual Convention of the International Communication Association*, 1975, Chicago, IL, USA.

DePalma, D. A. (2006). Quantifying the return on localization investment. In: Dunne, K. J. (ed.), *Perspectives on Localization*, American Translators Association Scholarly Monograph Series XIII, Amsterdam, The Netherlands: John Benjamins Pub. Co, p.15–36.

Detweiler, M. (2007). Managing UCD Within Agile Projects. *Interactions*, May / June 2007, p.40–42.

Dohler, P. N. (1997). Facets of Software Localization - A Translator's View. *Translation Journal*, 1 (1). [Online]. Available at: http://translationjournal.net/journal/softloc.htm [Accessed: 14 October 2015].

Donnellon, A. (1993). Crossfunctional Teams in Product Development: Accommodating the Structure to the Process. *Journal of Product Innovation Management*, 10 (5), p.377–392.

Dr. International. (2003). *Developing International Software*. 2nd ed. Redmond, WA, USA: Microsoft Press.

Dröge, R., Nowak, P. and Weber, T. (2006). *Programmieren mit dem .NET Compact Framework: Anwendungsentwicklung für mobile Geräte [Programming for the .NET Compact Framework: Application Development for Mobile Devices]*. Köln, Germany: Microsoft Press.

Dunne, K. J. (2006). A Copernican revolution. In: Dunne, K. J. (ed.), *Perspectives on Localization*, American Translators Association Scholarly Monograph Series XIII, Amsterdam, The Netherlands: John Benjamins Pub. Co, p.1–11.

Dunne, K. J. (2011). Managing the fourth dimension - Time and schedule in translation and localization projects. In: Dunne, K. J. and Dunne, E. S. (eds.), *Translation and Localization Project Management: The Art of the Possible*, American Translators

Association Scholarly Monograph Series XVI, Amsterdam, The Netherlands: John Benjamins Pub. Co, p.349–378.

Durkheim, E. (1893). *The division of labour in society*. New York, NY, USA.

Edwards, K. (2012). Inclusion and exclusion. *MultiLingual*, 23 (7), p.12–13.

Ellis, G. and Silk, J. (2014). Defend the integrity of physics. *Nature*, 516 (7531), p.321–323.

Elsen, H. (2005). Maschinelle Übersetzung in der Softwarelokalisierung [Machine translation in software localisation]. In: Reineke, D. and Schmitz, K.-D. (eds.), *Einführung in die Softwarelokalisierung [Introduction to software localisation]*, Tübingen, Germany: Narr, p.89–99.

Espinosa, A., Kraut, R., Slaughter, S., Lerch, J., Herbsleb, J. and Mockus, A. (2002). Shared Mental Models, Familiarity, and Coordination: A Multi-Method Study of Distributed Software Teams. *ICIS 2002 Proceedings*. [Online]. Available at: http://aisel.aisnet.org/icis2002/39 [Accessed: 14 October 2015].

Esselink, B. (2000). *A practical guide to localization*. Revised ed. Philadelphia, PA, USA: John Benjamins Pub. Co.

Esselink, B. (2003). Content Management and Translation - Two Worlds Together? In: Evers, V., Röse, K., Honold, P., Coronado, J. and Day, D. L. (eds.), *Designing for Global Markets 5 - Proceedings of the Fifth International Workshop on Internationalization of Products and Systems*, 2003, Berlin, Germany, p.3–5.

Esselink, B. (2006). The evolution of localization. In: Pym, A., Perekrestenko, A. and Starink, B. (eds.), *Translation Technology and its Teaching*, 14, Tarragona, Spain: Servei de Publicacions, p.21–29. [Online]. Available at: http://isg.urv.es/library/papers/isgbook.pdf [Accessed: 14 October 2015].

Evers, V., Maldonado, H., Brodecki, T. and Hinds, P. (2008). Relational vs. Group Self-Construal: Untangling the Role of National Culture in HRI. In: *3rd ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2008, Amsterdam, The Netherlands: IEEE, p.255–262.

Exton, C., Spillane, B. and Buckley, J. (2010). A Micro-Crowdsourcing implementation: the Babel software project. *Localisation Focus*, 9 (1), p.46–62. [Online]. Available at: http://www.localisation.ie/oldwebsite/resources/lfresearch/Vol9_1ExtonSpillaneBuckley.htm [Accessed: 14 October 2015].

Facebook. (2008). Facebook Releases Site in Spanish; German and French to Follow. *facebook press releases*. [Online]. Available at: http://newsroom.fb.com/news/2008/02/facebook-releases-site-in-spanish-german-and-french-to-follow/ [Accessed: 14 October 2015].

Ferreira, J. (2011). *User Experience Design and Agile Development: Integration as an on-going achievement in practice*. PhD thesis, Milton Keynes, UK: Open University.

Fetzer, J. H. (1988). Program verification: The very idea. *Communications of the ACM*, 31 (9), p.1048–1063.

Field, A. P. (2005). *Discovering statistics using SPSS*. 2nd ed. London, UK: SAGE.

Fissgus, U. and Seewald-Heg, U. (2005). Ausbildung in Softwarelokalisierung [Education in software localisation]. In: Reineke, D. and Schmitz, K.-D. (eds.), *Einführung in die Softwarelokalisierung [Introduction to software localisation]*, Tübingen, Germany: Narr, p.189–204.

Flick, U. (2002). *An introduction to qualitative research*. London, UK: Sage.

Forssell, D. (2001). One Translator's Thoughts on Software Localization. *Translation Journal*, 5 (3). [Online]. Available at: http://translationjournal.net/journal/17softloc.htm [Accessed: 14 October 2015].

Freigang, K.-H. (2000). Anforderungen an Hardware und Software [Requirements on Hardware and Software]. In: Schmitz, K.-D. and Wahle, K. (eds.), *Softwarelokalisierung [Software localisation]*, Tübingen, Germany: Stauffenburg, p.181–183.

Freigang, K.-H. and Reinke, U. (2005). Translation-Memory-Systeme in der Softwarelokalisierung [Translation memory systems in software localisation]. In: Reineke, D. and Schmitz, K.-D. (eds.), *Einführung in die Softwarelokalisierung [Introduction to software localisation]*, Tübingen, Germany: Narr, p.55–71.

French, J. R. P. and Raven, B. (1959). The bases of social power. In: Cartwright, D. (ed.), *Studies in Social Power*, Ann Arbour, MI, USA: Institute for Social Research, p.150–167.

Friedman, A. L. (1993). The Information Technology Field: An historical analysis. In: Quintas, P. (ed.), *Social dimensions of systems engineering - People, processes, policies and software development*, Ellis Horwood series in interactive information systems, New York, NY, USA: Ellis Horwood, p.18–33.

Fukuda, H. and Ohashi, Y. (1997). A Guideline for Reporting Results of Statistical Analysis in Japanese Journal of Clinical Oncology. *Japanese Journal of Clinical Oncology*, 27 (3), p.121–127.

GALA. (2015). Globalization and Localization Association. *Globalization and Localization Association*. [Online]. Available at: http://www.gala-global.org/ [Accessed: 14 October 2015].

del Galdo, E. M. (1996). Culture and Design. In: del Galdo, E. M. and Nielsen, J. (eds.), *International User Interfaces*, New York, NY, USA: John Wiley & Sons, p.74–87.

del Galdo, E. M. and Nielsen, J. (1996). Preface. In: del Galdo, E. M. and Nielsen, J. (eds.), *International User Interfaces*, New York, NY, USA: John Wiley & Sons, p.v – vii.

Gerth, H. and Mills, C. W. (1991). *From Max Weber: Essays in Sociology*. Oxon, United Kingdom: Routledge.

Giammarresi, S. (2011). Strategic views on localization project management - The importance of global product management and portfolio management. In: Dunne, K. J. and Dunne, E. S. (eds.), *Translation and Localization Project Management: The Art of the Possible*, American Translators Association Scholarly Monograph Series XVI, Amsterdam, The Netherlands: John Benjamins Pub. Co, p.17–50.

Gizaw, S. (2014). An Empirically Derived Personalisation Framework for Technical Support. In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014, ACM.

Gladwell, M. (2015). The Engineer's Lament. *The New Yorker*. [Online]. Available at: http://www.newyorker.com/magazine/2015/05/04/the-engineers-lament [Accessed: 14 October 2015].

Glaser, B. G. (1978). *Theoretical Sensitivity: Advances in the Methodology of Grounded Theory*. Mill Valley, CA, USA: Sociology Press.

Glaser, B. G. and Strauss, A. L. (2009). *The Discovery of Grounded Theory: Strategies for Qualitative Research*. 7th ed. Piscataway, NJ, USA: Transaction Publishers.

Glass, R. L. (2002). *Facts and fallacies of software engineering*. Boston, MA, USA: Addison-Wesley Professional.

Glass, R. L. (2005). The Plot to Deskill Software Engineering. *Communications of the ACM*, 48 (11), p.21–24.

Glass, R. L., Ramesh, V. and Vessey, I. (2004). An analysis of research in computing disciplines. *Communications of the ACM*, 47 (6), p.89–94.

Goggins, S. P. and Mascaro, C. (2011). Social Media Discourse and Culture: A Proposal for Comparative Informatics Research. In: *2nd International Workshop on Comparative Informatics*, 2011, Copenhagen, Denmark. [Online]. Available at: http://seangoggins.net/sites/default/files/ComparativeInformaticsStudyProposal-FINAL-V3.pdf [Accessed: 14 October 2015].

Green, T. R. G. (1994). Why software engineers don't listen to what psychologists don't tell them anyway. In: *User-centred requirements for software engineering environments*, Springer, p.323–333.

Grigas, G. (2014). Designing Tablet Computer Keyboards for European Languages. *Localisation Focus*, 13 (1), p.16–26. [Online]. Available at: http://www.localisation.ie/locfocus/issues/13/1 [Accessed: 14 October 2015].

Grinter, R. E. (1995). Using a configuration management tool to coordinate software development. In: *Proceedings of the Conference on Organizational Computing Systems*, 1995, ACM, p.168–177.

Grinter, R. E. (1996a). *Understanding Dependencies: A Study of the Coordination Challenges in Software Development*. PhD thesis, Irvine, CA, USA: University of California. [Online]. Available at:

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.5130&rep=rep1&type=pdf [Accessed: 14 October 2015].

Grinter, R. E. (1996b). Understanding the Role of Configuration Management Systems in Software Development. In: *Conference Companion on Human Factors in Computing Systems*, 1996, Vancouver, Canada, p.39–40.

de Groot, A. D. (1969). *Methodology: Foundations of inference and research in the behavioral sciences*. The Hague, The Netherlands: Mouton.

de Groot, A. D. (2014). The Meaning of 'Significance' for Different Types of Research. *Acta Psychologica*, 148, p.188–194.

Guttman, L. L. (1974). The Basis for Scalogram Analysis. In: Maranell, G. M. (ed.), *Scaling: A Sourcebook for Behavioral Scientists*, Chicago, IL, USA: Aldine Pub. Co., p.142–171.

Hacker, W. (1986). *Arbeitspsychologie: Psychische Regulation von Arbeitstätigkeiten [Work psychology: Psychological Regulation of Work Activities]*. Berlin, GDR: Deutscher Verlag der Wissenschaften.

Hagen, J. U. (2013). *Confronting Mistakes - Lessons from the Aviation Industry When Dealing With Error*. Basingstoke, UK: Palgrave Macmillan.

Hall, E. T. (1959). *The Silent Language*. New York, NY, USA: Doubleday & Co.

Hall, E. T. (1966). *The Hidden Dimension*. 1st ed. New York, NY, USA: Doubleday & Co.

Hall, E. T. (1977). *Beyond Culture*. New York, NY, USA: Anchor Books.

Hall, P. (1998). Designing Code Tables with Application to Nepali. *Journal of Global Information Management*, 6 (4), p.5–14.

Hall, P. (2006). Localisation in Nepal. *Localisation Focus*, 5 (2), p.21–23. [Online]. Available at: http://www.localisation.ie/oldwebsite/resources/locfocus/issues/Jun2006.pdf [Accessed: 14 October 2015].

Hall, P. (2015). Computerised writing for small languages. *Journal of Specialised Translation*, 24, p.163–186. [Online]. Available at: http://www.jostrans.org/issue24/art_hall.pdf [Accessed: 14 October 2015].

Hall, P. A. V. (2000). Software Internationalization Architectures for Decision Support Systems. In: Kersten, G. E., Zbigniew, M. and Gar-On Yeh, A. (eds.), *Decision Support Systems for Sustainable Development*, New York, NY, USA: Kluwer Academic Publishers, p.291–304.

Hall, P. A. V. (2002). Bridging the Digital Divide, the Future of Localisation. *The Electronic Journal of Information Systems in Developing Countries*, 8 (1), p.1–9.

Hall, P. A. V. (2004). Localising Nations, saving languages: moving from Unicode to Language Engineering. In: *Proceedings of the 26th International Conference on Translating*

and the Computer, 2004, London, UK. [Online]. Available at: http://mt-archive.info/Aslib-2004-Hall.pdf [Accessed: 14 October 2015].

Hall, P., Bal, B. K., Dhakhwa, S. and Regmi, B. N. (2014). Issues in Encoding the Writing of Nepal's Languages. In: *Computational Linguistics and Intelligent Text Processing*, Lecture Notes in Computer Science 8403, Berlin, Germany: Springer, p.52–67.

Hall, P., Ghimire, G. and Newton, M. (2009). Why Don't People Use Nepali Language Software? *Information Technologies and International Development*, 5 (1), p.65–79.

Hall, P., Lawson, C. and Minocha, S. (2003). Design Patterns as a Guide to the Cultural Localisation of Software. In: Evers, V., Röse, K., Honold, P., Coronado, J. and Day, D. L. (eds.), *Designing for Global Markets 5 - Proceedings of the Fifth International Workshop on Internationalization of Products and Systems*, 2003, Berlin, Germany, p.79–88.

Hall, P., Parikh, T., Minocha, S. and Venkatesh, H. (2002). Localisation in South Asia. *Localisation Focus*, 1 (3), p.4–5. [Online]. Available at: http://www.localisation.ie/oldwebsite/resources/locfocus/issues/2002dec.zip [Accessed: 14 October 2015].

Hammersley, M. (1992). *What's wrong with Ethnography?* New York, NY, USA: Routledge.

Haralambos, M., Holborn, M. and Heald, R. (2004). *Sociology: Themes and Perspectives*. 6th ed. London, UK: HarperCollins.

Haralambos, M., Holborn, M. and Heald, R. (2013). *Sociology: Themes and Perspectives*. 8th ed. London, UK: HarperCollins.

Harper, R., Rodden, T., Rogers, Y. and Sellen, A. (eds.). (2008). *Being Human: Human-Computer Interaction in the Year 2020*. Cambridge, UK: Microsoft Research.

Hartley, T. (2009). Technology and translation. In: Munday, J. (ed.), *The Routledge Companion to Translation Studies*, London, UK: Routledge, p.106–127.

Heath, H. and Cowley, S. (2004). Developing a grounded theory approach: a comparison of Glaser and Strauss. *International Journal of Nursing Studies*, 41 (2), p.141–150.

Herbsleb, J. D., Mockus, A., Finholt, T. A. and Grinter, R. E. (2000). Distance, dependencies, and delay in a global collaboration. In: *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, 2000, ACM, p.319–328.

Herbsleb, J. D., Mockus, A., Finholt, T. A. and Grinter, R. E. (2001). An empirical study of global software development: distance and speed. In: *Proceedings of the 23rd International Conference on Software Engineering*, 2001, IEEE Computer Society, p.81–90.

He, Z., Bustard, D. W. and Liu, X. (2002). Software internationalisation and localisation: practice and evolution. In: *Proceedings of the inaugural conference on the Principles and Practice of Programming and the second Workshop on Intermediate Representation Engineering for Virtual Machines*, 2002, National University of Ireland, p.89–94.

Hirschheim, R. and Klein, H. K. (1989). Four Paradigms of Information Systems Development. *Communications of the ACM*, 32 (10), p.1199–1216.

Hoda, R. (2011). *Self-Organizing Agile Teams: A Grounded Theory*. PhD thesis, Wellington, New Zealand: Victoria University of Wellington. [Online]. Available at: http://researcharchive.vuw.ac.nz/handle/10063/1617 [Accessed: 14 October 2015].

Hoda, R., Noble, J. and Marshall, S. (2010). Using Grounded Theory to Study the Human Aspects of Software Engineering. In: *HAoSE'10 Human Aspects of Software Engineering*, 2010, Reno, NV, USA: ACM.

Hoda, R., Noble, J. and Marshall, S. (2011). Grounded Theory for Geeks. In: *Proceedings of the 18th Conference on Pattern Languages of Programs PLoP'11*, 2011, Portland, OR, USA: ACM, p.24–41.

Hoda, R., Noble, J. and Marshall, S. (2012). Developing a grounded theory to explain the practices of self-organizing Agile teams. *Empirical Software Engineering*, 17 (6), p.609–639.

Hofstede, G. and Hofstede, G. J. (2005). *Cultures and organizations : software of the mind*. 2nd ed. New York, NY, USA: McGraw-Hill.

Hoft, N. L. (1996). Developing a cultural model. In: del Galdo, E. M. and Nielsen, J. (eds.), *International User Interfaces*, New York, NY, USA: John Wiley & Sons, p.41–73.

Hogan, J. M., Ho-Stuart, C. and Pham, B. (2004). Key Challenges in Software Internationalisation. In: Hogan, J., Montague, P., Purvis, M. and Steketee, C. (eds.), *Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation*, 32, 2004, Dunedin, New Zealand: Australian Computer Society, Inc., p.187–194.

Honkela, T., Lehtola, A., Kalliomäki, S., Suitiala, R., Hudson, R., Karkaletsis, V. and Vouros, G. (1997). A Recommended Globalization Method. In: Hall, P. A. V. and Hudson, R. (eds.), *Software Without Frontiers*, Wiley Series in Software Engineering Practice, Chichester, UK: John Wiley & Sons, p.33–49.

Hua, Z., Taslim, M. and Keating, M. (2014). Same but not the same: how much do we look into it for localization design? In: *Proceedings of HCI Korea 2015*, 2014, Seoul, Korea: Hanbit Media, Inc., p.83–87.

Hubscher-Davidson, S. E. (2009). Personal diversity and diverse personalities in translation: A study of individual differences. *Perspectives: Studies in translatology*, 17 (3), p.175–192.

Hudson, R. (1997). Introduction. In: Hall, P. A. V. and Hudson, R. (eds.), *Software Without Frontiers*, Wiley Series in Software Engineering Practice, Chichester, UK: John Wiley & Sons, p.1–13.

Hudson, R., McHugh, N. and Kalpakas, C. (1997). How Major Software Companies Approach Globalization. In: Hall, P. A. V. and Hudson, R. (eds.), *Software Without*

Frontiers, Wiley Series in Software Engineering Practice, Chichester, UK: John Wiley & Sons, p.15–32.

Illmensee, T. and Muff, A. (2009). 5 Users Every Friday: A Case Study in Applied Research. In: *AGILE '09 Proceedings of the 2009 Agile Conference*, 2009, Chicago, IL, USA, p.404–409.

Immonen, J. and Sajaniemi, J. (2003a). Globalisation Practices in the Finnish Software Industry. In: Evers, V., Röse, K., Honold, P., Coronado, J. and Day, D. L. (eds.), *Designing for Global Markets 5 - Proceedings of the Fifth International Workshop on Internationalization of Products and Systems*, 2003, Berlin, Germany, p.155–166.

Immonen, J. and Sajaniemi, J. (2003b). *Software Globalisation in Finland: A State-of-the-practice Survey*. Joensuu, Finland: University of Joensuu, Department of Computer Science. [Online]. Available at: ftp://cs.uef.fi/pub/Reports/A-2003-1.pdf [Accessed: 14 October 2015].

Irmler, U. and Hartwig, D. (2000). Sprachliche Qualität im lokalisierten Softwareprodukt: Kriterien und Vorgehensweisen der Qualitätssicherung [Linguistic quality in the localised software product: criterias and procedure of quality assurance]. In: Schmitz, K.-D. and Wahle, K. (eds.), *Softwarelokalisierung [Software localisation]*, Tübingen, Germany: Stauffenburg, p.89–100.

Isa, W. A. R. W. M., Noor, N. L. M. and Mehad, S. (2010). Web Architectural-Inducing Model (WA-IM) for Information Architecture in Cultural Context: An Empirical Investigation. *Journal of Digital Information Management*, 8 (5), p.330–337.

ISO/IEC JTC 1/SC 2. (1998). *ISO/IEC 8859-1:1998 8-bit single-byte coded graphic character sets - Part 1: Latin alphabet No.1*. Geneva, Switzerland: ISO/IEC.

ISO/IEC JTC 1/SC 2. (1999a). *ISO/IEC 8859-2:1999 8-bit single-byte coded graphic character sets - Part 2: Latin alphabet No.2*. Geneva, Switzerland: ISO/IEC.

ISO/IEC JTC 1/SC 2. (1999b). *ISO/IEC 8859-3:1999 8-bit single-byte coded graphic character sets - Part 3: Latin alphabet No.3*. Geneva, Switzerland: ISO/IEC.

ISO/IEC JTC 1/SC 2. (2011). *ISO/IEC 14651:2011(E) Information technology - International string ordering and comparison - Method for comparing character strings and description of the common template tailorable ordering*. Geneva, Switzerland: ISO/IEC.

ISO/IEC JTC 1/SC 2. (2014). *ISO/IEC 10646:2014 Universal Coded Character Set (UCS)*. Geneva, Switzerland: ISO/IEC.

ISO/IEC JTC 1/SC 7. (1991). *ISO/IEC 9126:1991 Software engineering - Product quality*. Geneva, Switzerland: ISO/IEC.

ISO/IEC JTC 1/SC 7. (2011). *ISO/IEC 25010:2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*. Geneva, Switzerland: ISO/IEC.

ISO/IEC JTC 1/SC 22. (1998). *ISO/IEC TR 11017:1998(E) Information technology - Framework for internationalization*. Geneva, Switzerland: ISO/IEC.

ISO/IEC JTC 1/SC 35. (2014). *ISO/IEC TR 30112 Information technology - Specification methods for cultural conventions*. Geneva, Switzerland: ISO/IEC.

ISO TC 37/SC 2. (2002). *ISO 639-1:2002 Code for the representation of names of languages*. Geneva, Switzerland: ISO.

ISO TC 46. (2013). *ISO 3166-1:2013 Codes for the representation of names of countries and their subdivisions - Part 1: Country codes*. Geneva, Switzerland: ISO.

Ito, M. and Nakakoji, K. (1996). Impact of Culture on User Interface Design. In: del Galdo, E. M. and Nielsen, J. (eds.), *International User Interfaces*, New York, NY, USA: John Wiley & Sons, p.105–126.

Jensen, M. C. and Meckling, W. H. (1976). Theory of the Firm: Managerial Behavior, Agency Costs and Ownership Structure. *Journal of financial economics*, 3 (4), p.305–360. [Online]. Available at: http://www.sfu.ca/~wainwrig/Econ400/jensen-meckling.pdf [Accessed: 14 October 2015].

Jin, L. (1997). Building an Internationalized Word Processor - A Case Study. In: Hall, P. A. V. and Hudson, R. (eds.), *Software Without Frontiers*, Wiley Series in Software Engineering Practice, Chichester, UK: John Wiley & Sons, p.123–133.

Juric, R., Kim, I. and Kuljis, J. (2003). Cross cultural web design: an experiences of developing UK and Korean cultural markers. In: *Proceedings of the 25th International Conference on Information Technology Interfaces 2003, ITI 2003*, 2003, Cavtat, Croatia: IEEE, p.309–313.

Kahler, T. (2000). Projektmanagement in der Softwarelokalisierung: Eine Einführung [Project management in software localisation: an introduction]. In: Schmitz, K.-D. and Wahle, K. (eds.), *Softwarelokalisierung [Software localisation]*, Tübingen, Germany: Stauffenburg, p.11–19.

Kalliomäki, S., Lehtola, A. and Lagus, K. (1997). Internationalization Analysis. In: Hall, P. A. V. and Hudson, R. (eds.), *Software Without Frontiers*, Wiley Series in Software Engineering Practice, Chichester, UK: John Wiley & Sons, p.51–82.

Kamppuri, M. (2011). *Theoretical and methodological challenges of cross-cultural interaction design*. PhD thesis, Joensuu, Finland: University of Eastern Finland. [Online]. Available at: http://epublications.uef.fi/pub/urn_isbn_978-952-61-0407-2/urn_isbn_978-952-61-0407-2.pdf [Accessed: 14 October 2015].

Karkaletsis, E. A., Spyropoulos, C. D. and Vouros, G. (1995). The Use of Terminological Knowledge Bases in Software Localisation. *Lecture Notes in Computer Science*, 898, p.175–188.

Khodadady, E. and Ghahari, S. (2012). Exploring the Relationship Between Foreign Language Proficiency and Cultural Intelligence. *International Journal of Language*

Learning and Applied Linguistics World, 1 (1), p.22–30. [Online]. Available at: http://profdoc.um.ac.ir/paper-abstract-1035942.html [Accessed: 14 October 2015].

Kim, B. Y. and Kang, B.-K. (2008). Cross-Functional Cooperation with Design Teams in New Product Development. *International Journal of Design*, 2 (3), p.43–54.

Kinzie, M. B., Delcourt, M. A. B. and Powers, S. M. (1994). Computer technologies: Attitudes and self-efficacy across undergraduate disciplines. *Research in Higher Education*, 35 (6), p.745–768.

Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C. and Rosenberg, J. (2008). Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Transactions on Software Engineering*, 28 (8), p.721–734.

Klein, J. T. (1990). *Interdisciplinarity: History, theory, and practice*. Detroit, MI, USA: Wayne State University Press.

Klein, J. T. (2005). Interdisciplinary Teamwork: The Dynamics of Collaboration and Integration. In: Derry, S. J., Schunn, C. D. and Gernsbacher, M. A. (eds.), *Interdisciplinary Collaboration: An Emerging Cognitive Science*, New York, NY, USA: Psychology Press, p.23–50.

Kling, R. (1996a). A Reader's Guide to Computerization and Controversy'. In: Kling, R. (ed.), *Computerization and Controversy: Value Conflicts and Social Choices*, 2nd ed., San Diego, CA, USA: Morgan Kaufmann, p.4–9.

Kling, R. (1996b). Where are the Payoffs from Computerization? Technology, Learning, and Organizational Change. In: Kling, R. (ed.), *Computerization and Controversy: Value Conflicts and Social Choices*, 2nd ed., San Diego, CA, USA: Morgan Kaufmann, p.239–260.

Kokkotos, S. and Spyropoulos, C. (1997a). An Architecture for Internationalization. In: Hall, P. A. V. and Hudson, R. (eds.), *Software Without Frontiers*, Wiley Series in Software Engineering Practice, Chichester, UK: John Wiley & Sons, p.111–122.

Kokkotos, S. and Spyropoulos, C. D. (1997b). An Architecture for Designing Internationalized Software. In: Budgen, D., Hoffnagle, G. and Trienekens, J. (eds.), *Eigth IEEE International Workshop on Software Technology and Engineering Practice incorporating Computer Aided Software Engineering*, 1997, London, UK: IEEE Computer Society, p.13–21.

Kokkotos, S., Spyropoulos, C., Honkela, T., Kapylä, T., Lagus, K. and Hall, P. A. V. (1997). Languages and Character Sets. In: Hall, P. A. V. and Hudson, R. (eds.), *Software Without Frontiers*, Wiley Series in Software Engineering Practice, Chichester, UK: John Wiley & Sons, p.135–157.

Kroeber, A. L., Kluckhohn, C., Untereiner, W. and Meyer, A. G. (1952). *Culture - A critical review of concepts and definitions*. Cambridge, MA, USA: Peabody Museum of American Archaeology and Ethymology, Harvard University.

Kruchten, P. (2005). Casting Software Design in the Function-Behavior-Structure Framework. *IEEE Software*, 22 (2), p.52–58.

Kumhyr, D., Merrill, C. and Spalink, K. (1994). Internationalization and Translatability. In: *Proceedings of the First Conference of the Association for Machine Translation in the Americas*, 1, 1994, p.142–148.

Kvale, S. (2007). *Doing Interviews*, Sage Qualitative Research Kit 2. London, UK: Sage.

Lagus, K., Suitiala, R. and Honkela, T. (1997). Culture, Conventions and Local Practices. In: Hall, P. A. V. and Hudson, R. (eds.), *Software Without Frontiers*, Wiley Series in Software Engineering Practice, Chichester, UK: John Wiley & Sons, p.159–166.

Landsberger, H. A. (1958). *Hawthorne Revisited*. Ithaca, NY, USA: New York State School of Industrial and Labor Relations.

Larman, C. and Basili, V. R. (2003). Iterative and incremental development: A brief history. *Computer*, 36 (6), p.47–56.

Latane, B., Williams, K. and Harkins, S. (1979). Many Hands Make Light the Work: The Causes and Consequences of Social Loafing. *Journal of Personality and Social Psychology*, 37 (6), p.822–832.

Law, L.-C. (2003). Challenges in Software Localization: A Case Study of a European Educational Network. In: Evers, V., Röse, K., Honold, P., Coronado, J. and Day, D. L. (eds.), *Designing for Global Markets 5 - Proceedings of the Fifth International Workshop on Internationalization of Products and Systems*, 2003, Berlin, Germany, p.267–279.

LeBlanc, M. (2013). Translators on translation memory (TM). Results of an ethnographic study in three translation services and agencies. *Translation & Interpreting*, 5 (2), p.1–13. [Online]. Available at: http://trans-int.org/index.php/transint/article/view/228 [Accessed: 14 October 2015].

Leedy, P. D. and Ormrod, J. E. (2013). *Practical Research: Planning and Design*. 10th ed. Upper Saddle River, NJ, USA: Pearson Education.

Lehtola, A., Kalliomäki, S., Honkela, T. and Lillqvist, R. (1997). An Application Framework for Internationalization. In: Hall, P. A. V. and Hudson, R. (eds.), *Software Without Frontiers*, Wiley Series in Software Engineering Practice, Chichester, UK: John Wiley & Sons, p.83–110.

Leith, P. (1986). Fundamental errors in legal logic programming. *The Computer Journal*, 29 (6), p.545–552.

Lenker, M., Anastasiou, D. and Buckley, J. (2011). Workflow Specification for Enterprise Localisation. *Localisation Focus*, 9 (1), p.26–35. [Online]. Available at: http://www.localisation.ie/oldwebsite/resources/lfresearch/Vol9_1LenkerAnastasiouBuckley.htm [Accessed: 14 October 2015].

Levy, D. (2007). *Love and sex with Robots: the evolution of human-robot relationships*. New York, NY, USA: HarperCollins.

Lewis, D., Curran, S., Feeney, K., Etzioni, Z., Keeney, J., Way, A. and Schäler, R. (2009). Web Service Integration for Next Generation Localisation. In: *Proceedings of the Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, SETQA-NLP '09, 2009, Stroudsburg, PA, USA: Association for Computational Linguistics, p.47–55.

Liem, A., Vatrapu, R. and Clemmensen, T. (2011). A Culture and Touchpoint Approach to Improve Experiences in Service and Human-Computer Interaction Design. In: *2nd International Workshop on Comparative Informatics*, 2011, Copenhagen, Denmark.

Linberg, K. R. (1999). Software developer perceptions about software project failure: a case study. *Journal of Systems and Software*, 49 (2), p.177–192.

Lindvall, M. and Rus, I. (2000). Process Diversity in Software Development. *IEEE Software*, 17 (4), p.14–18.

Lingotek. (2011). Lingotek - the translation network. *The Localization Industry Standards Association (LISA) shuts down operations*. [Online]. Available at: http://www.lingotek.com/localization-industry-standards-association-lisa-shuts-down-operations [Accessed: 14 October 2015].

Linna, P. and Jaakkola, H. (2010). Toward finding culture assessment tools for SE companies. In: *Proceedings of the PICMET '10 Conference*, 18 July 2010, Phuket, Thailand, p.1–6.

Liu, Z. and Zhang, G. (2011). Exploring Culture Factors in HCI Design: A Perspective from SEUC. In: *Proceedings of the 2nd International Workshop on Comparative Informatics (IWCI 2011)*, 2011, Copenhagen, Denmark.

Low, J., Johnson, J., Hall, P., Hovenden, F., Rachel, J., Robinson, H. and Woolgar, S. (1996). Read this and change the way you feel about software engineering. *Information and Software Technology*, 38 (2), p.77–87.

Mahemoff, M. J. and Johnston, L. J. (1998). Software Internationalisation: Implications for Requirements Engineering. In: Fowler, D. and Dawson, L. (eds.), *Proceedings of the Third Australian Workshop on Requirements Engineering*, 1998, Geelong, Australia, p.83–90.

Malmkjaer, K. S. (2008). Translation competence and the aesthetic attitude. In: Pym, A., Shlesinger, M. and Simeoni (eds.), *Beyond Descriptive Translation Studies*, Benjamins Translation Library 75, Amsterdam, The Netherlands: John Benjamins Pub. Co, p.293–309.

Marcus, A. (1996). Icon and Symbol Design Issues for Graphical User Interfaces. In: del Galdo, E. M. and Nielsen, J. (eds.), *International User Interfaces*, John Wiley & Sons, p.257–270.

Marcus, A. and Gould, E. W. (2000). Crosscurrents: Cultural Dimensions and Global Web User-Interface Design. *ACM Interactions*, 7 (4), p.32–46.

Martin, A. M. (2009). *The Role of Customers in Extreme Programming Projects*. PhD thesis, Wellington, New Zealand: Victoria University of Wellington. [Online]. Available at: http://researcharchive.vuw.ac.nz/handle/10063/877 [Accessed: 14 October 2015].

Matsumoto, D., LeRoux, J., Ratzlaff, C., Tatani, H., Uchida, H., Kim, C. and Araki, S. (2001). Development and validation of a measure of intercultural adjustment potential in Japanese sojourners: The Intercultural Adjustment Potential Scale (ICAPS). *International Journal of Intercultural Relations*, 25 (5), p.483–510.

Maxwell, K. (2002). The Maturation of HCI: Moving beyond Usability toward Holistic Interaction. In: Carroll, J. M. (ed.), *Human-Computer Interaction in the New Millenium*, New York, NY, USA: ACM Press, p.191–209.

Maylor, H. (2010). *Project Management*. 4th ed. New York, NY, USA: Pearson Education.

McConnell, S. (2004). *Code Complete*. 2nd ed. Redmond, WA, USA: Microsoft Press.

McHugh, N., Honkela, T. and Hudson, R. (1997). Quality Assurance. In: Hall, P. A. V. and Hudson, R. (eds.), *Software Without Frontiers*, Wiley Series in Software Engineering Practice, Chichester, UK: John Wiley & Sons, p.219–228.

McKethan, K. A. and White, G. (2005). Demystifying Software Globalization. *Translation Journal*, 9 (2). [Online]. Available at: http://accurapid.com/journal/32global.htm [Accessed: 14 October 2015].

McSweeney, B. (2002). Hofstede's model of national cultural differences and their consequences: A triumph of faith-a failure of analysis. *Human relations*, 55 (1), p.89–118.

Microsoft Corporation Editorial Style Board. (2004). *Microsoft Manual of Style for Technical Publications*. 3rd ed. Redmond, WA, USA: Microsoft Press.

Milder, M. (2000). Projektmanagement in einem globalen Umfeld [Project management in a global environment]. In: Schmitz, K.-D. and Wahle, K. (eds.), *Softwarelokalisierung [Software localisation]*, Tübingen, Germany: Stauffenburg, p.21–30.

Miles, M. B. and Huberman, A. M. (1994). *Qualitative Data Analysis: An expanded sourcebook*. 2nd ed. London, UK: Sage.

Milgram, S. (1963). Behavioral study of obedience. *The Journal of Abnormal and Social Psychology*, 67 (4), p.371–378.

Milgram, S. (1974). *Obedience to Authority: An Experimental View*. London, UK: HarperCollins.

Moorkens, J. (2011). Translation Memories guarantee consistency: Truth or fiction? In: *Proceedings of the 33. International Conference on Translating and the Computer*, 2011, London, UK. [Online]. Available at: http://www.mt-archive.info/Aslib-2011-Moorkens.pdf [Accessed: 14 October 2015].

Moorkens, J. (2012a). A mixed-methods study of consistency in translation memories. *Localisation Focus*, 11 (1), p.14–26. [Online]. Available at: http://www.localisation.ie/oldwebsite/resources/lfresearch/Vol11_1Moorkens.htm [Accessed: 14 October 2015].

Moorkens, J. (2012b). *Measuring Consistency in Translation Memories: a Mixed-Methods Case Study*. PhD thesis, Dublin, Ireland: Dublin City University. [Online]. Available at: http://www.cngl.ie/wp-content/uploads/2014/10/Josh-Moorkens_PhD_Thesis.pdf [Accessed: 14 October 2015].

Morado Vásquez, L., Anastasiou, D., Exton, C. and O'Keefe, I. (2011). Web 2.0 and Localisation. In: *Proceedings of the Social Media Engagement Workshop, World Wide Web Conference*, 28 March 2011, Hydarabad, India.

Morado Vázquez, L. and Mooney, S. (2010). XLIFF Phoenix and LMC Builder: Organising, capturing and using localisation data and metadata. In: *The Annual Conference Proceedings of the Localisation Research Centre, Brave New World, University of Limerick, LRC XV*, 2010, Limerick, Ireland.

Mor, S., Toma, C., Schweinsberg, M. and Ames, D. (2015). Intercultural Judgment Accuracy and the Role of Social Projection Processes. *Centre Emile Bernheim Working Papers*, 15 (29). [Online]. Available at: https://dipot.ulb.ac.be/dspace/bitstream/2013/206240/3/wp15029.pdf [Accessed: 14 October 2015].

Munday, J. (2009). Issues in translation studies. In: Munday, J. (ed.), *The Routledge Companion to Translation Studies*, London, UK: Routledge, p.1–19.

Nardi, B. (2011). Occupational Identities. In: *2nd International Workshop on Comparative Informatics*, December 2011, Copenhagen, Denmark.

Nardi, B., Vatrapu, R. and Clemmensen, T. (2011). Comparative Informatics. *Interactions*, 18 (2), p.28–33.

Nielsen, J. (1996a). International Usability Engineering. In: del Galdo, E. M. and Nielsen, J. (eds.), *International User Interfaces*, New York, NY, USA: John Wiley & Sons, p.1–13.

Nielsen, J. (1996b). International Usability Testing. *useit.com*. [Online]. Available at: http://www.useit.com/papers/international_usetest.html [Accessed: 14 October 2015].

Nissani, M. (1990). A cognitive reinterpretation of Stanley Milgram's observations on obedience to authority. *American Psychologist*, 45, p.1384–1385. [Online]. Available at: http://drnissani.net/MNISSANI/PAGEPUB/milgram.htm [Accessed: 14 October 2015].

Norušis, M. J. (2006). *SPSS 15.0 Guide to Data Analysis*. Upper Saddle River, NJ, USA: Prentice Hall.

Nuzzo, R. (2014). Statistical Errors. *Nature*, 506 (13), p.150–152.

O'Donnell, A. M., DuRussel, L. A. and Derry, S. J. (1997). *Cognitive processes in interdisciplinary groups: Problems and possibilities*. Research Monograph, National Institute for Science Education, University of Wisconsin-Madison.

O'Keeffe, I. R. (2009). Music Localisation: Active Music Content for Web Pages. *Localisation Focus*, 8 (1), p.67–81. [Online]. Available at: http://www.localisation.ie/oldwebsite/resources/lfresearch/Vol8_1OKeeffe.htm [Accessed: 14 October 2015].

Orlikowski, W. J. (1991). Integrated information environment or matrix of control? The contradictory implications of information technology. *Accounting, Management and Information Technologies*, 1 (1), p.9–42.

O'Sullivan, P. (2001a). Pat O'Sullivan's award winning localisation thesis covers new ground. *Localisation Ireland*, 5 (2), p.6–10. [Online]. Available at: http://www.localisation.ie/oldwebsite/resources/locfocus/issues/2001sept.pdf [Accessed: 14 October 2015].

O'Sullivan, P. and Hyland, M. (2004). Engineering Global Software - A System Test Standards Perspective. *Localisation Focus*, 3 (3), p.6–8. [Online]. Available at: http://www.localisation.ie/oldwebsite/resources/locfocus/issues/Sept2004.zip [Accessed: 14 October 2015].

O'Sullivan, P. J. M. (2001b). *A Paradigm for Creating Multilingual Interfaces*. PhD thesis, Limerick, Ireland: University of Limerick. [Online]. Available at: http://www.localisation.ie/oldwebsite/resources/Awards/Theses/PhD-PatOSullivan.zip [Accessed: 14 October 2015].

O'Sullivan, P., Wallace, M. and Yousuf, N. (2003). A Software Model Approach to Accommodating Cultural Diversity in the Development of Multilingual Applications. In: Evers, V., Röse, K., Honold, P., Coronado, J. and Day, D. L. (eds.), *Designing for Global Markets 5 - Proceedings of the Fifth International Workshop on Internationalization of Products and Systems*, 2003, Berlin, Germany, p.9–28.

O'Sullivan, S. (1989). Problems in software translation and how to avoid them. In: *Proceedings of the 11th International Conference on Translating and the Computer*, 1989, London, UK, p.20–23. [Online]. Available at: http://www.mt-archive.info/70/Aslib-1989-OSullivan.pdf [Accessed: 14 October 2015].

Ottmann, A. (2005). Lokalisierung von Softwareoberflächen [Localisation of software user interfaces]. In: Reineke, D. and Schmitz, K.-D. (eds.), *Einführung in die Softwarelokalisierung [Introduction to software localisation]*, Tübingen, Germany: Narr, p.101–115.

Paetsch, F., Eberlein, A. and Maurer, F. (2003). Requirements engineering and agile software development. In: *Proceedings of the 12th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2003, IEEE Computer Society, p.308–308.

Papaioannou, J. (2005). The Localisation Outsourcing Decision: How to. *Localisation Focus*, 4 (3), p.17–20. [Online]. Available at: http://www.localisation.ie/oldwebsite/resources/lfresearch/Vol4_3Papaioannou.htm [Accessed: 14 October 2015].

Peng, W., Yang, X. and Zhu, F. (2009). Automation technique of software internationalization and localization based on lexical analysis. In: *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, 403, 2009, Seoul, Korea: ACM, p.970–975.

Pérez-Quiñones, M. A., Padilla-Falto, O. I. and McDevitt, K. (2005). Automatic Language Translation for User Interfaces. In: *TAPIA'05 2005 Richard Tapia Celebration of Diversity in Computing Conference*, 2005, Albuquerque, NM, USA: ACM, p.60–63.

Perlman, G. (1999). Universal Web Access: Delivering Services to Everyone. In: *CHI'99 Extended Abstracts on Human Factors in Computing Systems*, 1999, Pittsburgh, PA, USA, p.347.

Perry, D. E., Staudenmayer, N. A. and Votta, L. G. (1996). Understanding and Improving Time Usage in Software Development. In: Fuggetta, A. and Wolf, A. (eds.), *Software Process*, Trends in Software 4, Chichester, UK: Wiley-Blackwell.

Perry, D., Porter, A. and Votta, L. (2000). Empirical Studies of Software Engineering: A Roadmap. In: *Proceedings of the Conference on The Future of Sofware Engineering*, 2000, Limerick, Ireland, p.345–355.

Pinto, J. K. (2015). *Project Management: Achieving Competitive Advantage*. 4th ed. Boston, MA, USA: Prentice Hall.

Pittenger, D. J. (1993). Measuring the MBTI… and coming up short. *Journal of Career Planning and Employment*, 54 (1), p.48–52. [Online]. Available at: http://www.indiana.edu/~jobtalk/Articles/develop/mbti.pdf [Accessed: 14 October 2015].

Plonka, L., Segal, J., Sharp, H. and van der Linden, J. (2011). Collaboration in Pair Programming: Driving and Switching. In: *Agile Processes in Software Engineering and Extreme Programming*, Lecture Notes in Business Information Processing 77, Berlin/Heidelberg, Germany: Springer, p.43–59.

Porsiel, J. (2008). Machine translation at Volkswagen: a case study. *MultiLingual*, 19 (8), p.58–61.

Portanieri, F. and Amara, F. (1996). Arabization of Graphical User Interfaces. In: del Galdo, E. M. and Nielsen, J. (eds.), *International User Interfaces*, John Wiley & Sons, p.127–150.

Pym, A. (2008). On Toury's laws of how translators translate. In: Pym, A., Shlesinger, M. and Simeoni, D. (eds.), *Beyond Descriptive Translation Studies*, Benjamins Translation Library 75, Amsterdam, The Netherlands: John Benjamins Pub. Co, p.324–328.

Quintas, P. (1993). Introduction - Living the Lifecycle: Social processes in software and systems development. In: Quintas, P. (ed.), *Social dimensions of systems engineering - People, processes, policies and software development*, Ellis Horwood series in interactive information systems, New York, NY, USA: Ellis Horwood, p.1–17.

Rafii, F. and Perkins, S. (1995). Internationalizing software with concurrent engineering. *IEEE Software*, 5 (12), p.39–46. [Online]. Available at: http://www.computer.org/csdl/mags/so/1995/05/s5039.pdf [Accessed: 14 October 2015].

Randall, D., Hughes, J. and Shapiro, D. (1993). Systems Development - The Fourth Dimension: Perspectives on the social organization of work. In: Quintas, P. (ed.), *Social dimensions of systems engineering - People, processes, policies and software development*, Ellis Horwood series in interactive information systems, New York, NY, USA: Ellis Horwood, p.197–214.

Rauterberg, M. (2006). Usability in the Future - explicit and implicit effects in cultural computing. In: Heinecke, A. M. and Paul, H. (eds.), *Mensch und Computer 2006: Mensch und Computer im Strukturwandel [Human and Computer 2006: Human and Computer within Structural Change]*, Oldenbourg Verlag, p.29–36.

Reineke, D. (2005). Softwarelokalisierungswerkzeuge [Software localisation tools]. In: Reineke, D. and Schmitz, K.-D. (eds.), *Einführung in die Softwarelokalisierung [Introduction to software localisation]*, Tübingen, Germany: Narr, p.73–87.

Ressin, M. (2012). Empirically Researching Development of International Software. In: *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, 2 June 2012, Zürich, Switzerland.

Rico, C. and Torrejón, E. (2012). Skills and Profile of the New Role of the Translator as MT Post-editor. *Tradumàtica*, 10, p.166–178. [Online]. Available at: http://ddd.uab.cat/record/105642/ [Accessed: 14 October 2015].

Robinson, H., Hall, P., Hovenden, F. and Rachel, J. (1998). Postmodern software development. *The Computer Journal*, 41 (6), p.363–375.

Robson, C. (2011). *Real World Research*. 3rd ed. Oxford, UK: Wiley-Blackwell.

Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14 (2), p.131–164.

Rusakevičienė, A. and Kriaučionytė, R. (2012). Translating the Baltic languages. *MultiLingual*, 23 (2), p.34–38.

Russo, P. and Boor, S. (1993). How Fluent is Your Interface? Designing for International Users. In: *INTERCHI'93*, 1993, Amsterdam, The Netherlands, p.342–347.

Ryan, L., Anastasiou, D. and Cleary, Y. (2009). Using Content Development Guidelines to Reduce the Cost of Localising Digital Content. *Localisation Focus*, 8 (1), p.11–28. [Online]. Available at:

http://www.localisation.ie/oldwebsite/resources/lfresearch/Vol8_1RyanAnastasiouCleary.htm [Accessed: 14 October 2015].

Sachse, F. (2005). Lokalisierungsformate [Localisation formats]. In: Reineke, D. and Schmitz, K.-D. (eds.), *Einführung in die Softwarelokalisierung [Introduction to software localisation]*, Tübingen, Germany: Narr, p.145–167.

Salo, O. and Abrahamsson, P. (2004). Empirical evaluation of agile software development: The controlled case study approach. In: Bomarius, F. and Iida, H. (eds.), *Product Focused Sofware Process Improvement*, Lecture Notes in Computer Science 3009, Berlin/Heidelberg, Germany: Springer, p.408–423.

Sasikumar, M. and Hegde, J. J. (2004). Software localisation: some issues and challenges. In: *SCALLA 2004*, 2004, Kathmandu, Nepal. [Online]. Available at: http://kbcs.in/downloads/papers/Localisation.pdf [Accessed: 14 October 2015].

Schäler, R. (1994). A Practical Evaluation of an Integrated Translation Tool during a Large Scale Localisation Project. In: *Proceedings of the 4th Conference on Applied Natural Language Processing*, 1994, Stuttgart, Germany.

Schäler, R. (2007). Reverse localisation. *Localisation Focus*, 6 (1), p.39–48. [Online]. Available at: http://www.localisation.ie/oldwebsite/resources/lfresearch/Vol6_1Schaler.htm [Accessed: 14 October 2015].

Schubert, K. (2009). Positioning translation in technical communication studies. *Journal of Specialised Translation*, 11, p.17–30. [Online]. Available at: http://www.jostrans.org/issue11/art_schubert.pdf [Accessed: 14 October 2015].

Seaman, C. B. (1999). Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Transactions on Software Engineering*, 25 (4), p.557–572.

Séguinot, C. (2007). Translation and the Changing Profession: A Cross-Disciplinary Perspective. *TTR: Traduction, terminologie, rédaction*, 20 (1), p.171–191. [Online]. Available at: http://www.erudit.org/revue/ttr/2007/v20/n1/018502ar.html [Accessed: 14 October 2015].

Sharp, H., Woodman, M. and Hovenden, F. (2005). Using metaphor to analyse qualitative data: Vulcans and humans in software development. *Empirical Software Engineering*, 10 (3), p.343–365.

Shneiderman, B., Mayer, R., McKay, D. and Heller, P. (1977). Experimental investigations of the utility of detailed flowcharts in programming. *Communications of the ACM*, 20 (6), p.373–381.

Sikes, R. (2011). Rethinking the role of the localization project manager. In: Dunne, K. J. and Dunne, E. S. (eds.), *Translation and Localization Project Management: The Art of the Possible*, American Translators Association Scholarly Monograph Series XVI, Amsterdam, The Netherlands: John Benjamins Pub. Co, p.235–264.

Smith, A. and Dunckley, L. (2007). Issues for Human-Computer Interaction in Developing Countries. In: *User Centered Design and International Development, Workshop at CHI 2007*, 2007, p.2.

Smith, A., Dunckley, L., French, T., Minocha, S. and Chang, Y. (2004). A process model for developing usable cross-cultural websites. *Interacting with Computers*, 16, p.63–91.

Smith, A., Joshi, A., Liu, Z., Bannon, L., Gulliksen, J. and Li, C. (2007). Institutionalizing HCI in Asia. In: *Human-Computer Interaction – INTERACT 2007*, Lecture Notes in Computer Science 4663, 2, Springer, p.85–99.

Smith, V. (2013). *Sociology of Work: An Encyclopedia*. Los Angeles, CA, USA: SAGE Publications.

Solheim, J. A. and Rowland, J. H. (1993). An empirical study of testing and integration strategies using artificial software systems. *IEEE Transactions on Software Engineering*, 19 (10), p.941–949.

Sommerville, I. and Dewsbury, G. (2007). Dependable domestic systems design: A socio-technical approach. *Interacting with Computers*, 19 (4), p.438–456.

Srivastava, A. and Thomson, S. B. (2009). Framework analysis: a qualitative methodology for applied policy research. *Journal of Administration & Governance*, 4 (2), p.72–79. [Online]. Available at: http://www.joaag.com/uploads/06_Research_Note_Srivastava_and_Thomson_4_2_.pdf [Accessed: 14 October 2015].

Stamey, J. W. and Speights, W. S. (1999). Website localization. In: *Proceedings of the 17th Annual International Conference on Computer Documentation*, 1999, New Orleans, LA, USA: ACM, p.127–130.

Stewart, O., Lubensky, D., Macdonald, S. and Marcotte, J. (2010). Using Machine Translation for the Localization of Electronic Support Content: Evaluating End-User Satisfaction. In: *9th Conference of the Association for Machine Translation in the Americas (AMTA)*, 2010, Denver, CO, USA. [Online]. Available at: http://www.mt-archive.info/AMTA-2010-Stewart.pdf [Accessed: 14 October 2015].

Stoeller, W. (2011). Global virtual teams. In: Dunne, K. J. and Dunne, E. S. (eds.), *Translation and Localization Project Management: The Art of the Possible*, American Translators Association Scholarly Monograph Series XVI, Amsterdam, The Netherlands: John Benjamins Pub. Co, p.289–317.

Strauss, A. and Corbin, J. M. (1998). *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Newbury Park, CA, USA: Sage.

Sturm, C. (2002). TLCC - Towards a framework for systematic and successful product internationalization. In: Coronado, J., Day, D. L. and Hall, B. (eds.), *Designing for Global Markets 4 - Proceedings of the 4th International Workshop on Internationalisation of Products and Systems*, 2002, Austin, TX, USA, p.61–70.

Sun, H. (2002). Why Cultural Contexts Are Missing - A Rhetorical Critique of Localization Practices. In: *Leading the Technical Communication Revolution - 49th Annual Conference of the Society for Technical Communication*, 2002, Nashville, TN, USA, p.164–168.

Sun, H. (2004a). *Expanding the Scope of Localization: A Cultural Usability Perspective on Mobile Text Messaging Use in American and Chinese Contexts*. PhD thesis, Troy, NY, USA: Rensselaer Polytechnic Institute. [Online]. Available at: http://www.localisation.ie/sites/default/files/best-thesis/sun_diss.pdf [Accessed: 14 October 2015].

Sun, H. (2004b). Understanding the Localisation Process of Mobile Text Messaging on a Cultural Circuit. *Localisation Focus*, 3 (4), p.9. [Online]. Available at: http://www.localisation.ie/oldwebsite/resources/locfocus/issues/Dec2004.pdf [Accessed: 14 October 2015].

Sy, D. (2007). Adapting Usability Investigations for Agile User-Centered Design. *Journal of Usability Studies*, 2 (3), p.112–132.

Szuki, A. (1988). Aptitudes of translators and interpreters. *Journal des traducteurs/Translators' Journal*, 33 (1), p.108–114. [Online]. Available at: http://www.erudit.org/revue/META/1988/v33/n1/004314ar.pdf [Accessed: 14 October 2015].

Tarquini, G., Nishio, N. and O'Keeffe, I. (2010). Localisation in the stream: Assessing GILT strategies in relation to online services - The case of JAL web site in Japanese and English. In: *2010 IEEE International Professional Communication Conference (IPCC 2010)*, 7 July 2010, Enschede, The Netherlands, p.350–356.

Thayer, A. and Kolko, B. E. (2004). Localization of digital games: The process of blending for the global games market. *Technical Communication*, 51 (4), p.477–488.

Thicke, L. (2012). Automating Intel's multilingual chat. *MultiLingual*, 23 (1), p.14–16.

Thomas, D. C., Stahl, G., Ravlin, E. C., Poelmans, S., Pekerti, A., Maznevski, M., Lazarova, M. B., Elon, E., Ekelund, B. Z., Cerdin, J.-L., Brislin, R., Aycan, Z. and Au, K. (2012). Development of the Cultural Intelligence Assessment. In: Mobley, W. H., Wang, Y. and Li, M. (eds.), *Advances in Global Leadership*, 7, 155 - 178, Bingley, UK: Emerald.

Thomas, K. W. (1992). Conflict and negotiation processes in organizations. In: Dunnette, D. and Hough, L. M. (eds.), *Handbook of industrial and organizational psychology*, 2nd ed., 3, Palo Alto, CA, USA: Consulting Psychologists Press, p.651–717.

Thompson, N. (2003). *Communication and Language: A handbook of theory and practice*. Basingstoke, UK: Palgrave Macmillan.

Tichy, W. F. (1982). Design, implementation, and evaluation of a revision control system. In: *Proceedings of the 6th International Conference on Software Engineering*, 1982, IEEE Computer Society Press, p.58–67.

Trist, E. and Murray, H. (1990). Historical overview: the foundation and development of the Tavistock Institute. In: Trist, E. and Murray, H. (eds.), *The social engagement of social science: A Tavistock anthology*, 1, Philadelphia, PA, USA: University of Pennsylvania Press, p.45–67.

Trompenaars, F. and Hampden-Turner, C. (1998). *Riding the waves of culture*. New York, NY, USA: McGraw-Hill.

Truex, D., Baskerville, R. and Travis, J. (2000). Amethodical systems development: the deferred meaning of systems development methods. *Accounting, management and information technologies*, 10 (1), p.53–79.

Tsvetkov, N. and Tsvetkov, V. (2011). Effective communication in translation and localization project management. In: Dunne, K. J. and Dunne, E. S. (eds.), *Translation and Localization Project Management: The Art of the Possible*, American Translators Association Scholarly Monograph Series XVI, Amsterdam, The Netherlands: John Benjamins Pub. Co, p.189–210.

Tubbs, S. L. and Moss, S. (2003). *Human Communication: Principles and Contexts*. 9th ed. Boston, MA, USA: McGraw-Hill.

Tuckman, B. W. (1965). Developmental sequence in small groups. *Psychological Bulletin*, 63 (6), p.384–399.

Tuffley, D. (2003). Improving Information Systems Usability by Having a Technical Writer Facilitate Communication when Developing the Requirements Specification. In: Evers, V., Röse, K., Honold, P., Coronado, J. and Day, D. L. (eds.), *Designing for Global Markets 5 - Proceedings of the Fifth International Workshop on Internationalization of Products and Systems*, 2003, Berlin, Germany, p.31–40.

Turk, D., France, R. and Rumpe, B. (2002). Limitations of Agile Software Processes. In: *Third International Conference on eXtreme Programming and Agile Processes in Software Engineering*, 2002, Alghero, Italy, p.43–46.

Umarji, M. and Seaman, C. (2005). Predicting Acceptance of Software Process Improvement. In: *Proceedings of the 2005 Workshop on Human and Social Factors of Software Engineering HSSE'05*, 2005, St. Louis, MS, USA.

United Nations. (2012). *Information Economy Report 2012 - The Software Industry and Developing Countries*. Geneva, Switzerland: United Nations Publication. [Online]. Available at: http://unctad.org/en/PublicationsLibrary/ier2012_en.pdf [Accessed: 14 October 2015].

United Nations. (2015a). United Nations Member States. *United Nations Member States*. [Online]. Available at: http://www.un.org/en/members/ [Accessed: 14 October 2015].

United Nations. (2015b). United Nations member States - Non-member state maintaining observer mission. *United Nations Non-member States*. [Online]. Available at: http://www.un.org/en/members/nonmembers.shtml [Accessed: 14 October 2015].

UWL. (2008). *Research Ethics Code of Practice*. London, UK: UWL. [Online]. Available at: http://www.uwl.ac.uk/sites/default/files/Departments/Research/Web/PDF/research_ethics_codes_of_practice.pdf [Accessed: 14 October 2015].

UWL. (2015). *Research Student Handbook AY 2015-2016 Appendix 1 Regulations for the Award of the University's Research Degrees*. London, UK: UWL.

Vasiļjevs, A. and Sāmīte, I. (2012). Machine translation for less-resourced languages. *MultiLingual*, 23 (1), p.25–30.

Vatrapu, R. K. (2011). Comparative Informatics: Cultural and Linguistic Influences in Computer Supported Collaboration. In: *2nd International Workshop on Comparative Informatics*, 2011, Copenhagen, Denmark.

Vinekar, V., Slinkman, C. W. and Nerur, S. (2006). Can Agile and Traditional Systems Development Approaches Coexist? An Ambidextrous View. *Information Systems Management*, 23 (3), p.31–42.

Vouros, G., Karkaletsis, V. and Spyropoulos, C. (1997). Documentation and Translation. In: Hall, P. A. V. and Hudson, R. (eds.), *Software Without Frontiers*, Wiley Series in Software Engineering Practice, Chichester, UK: John Wiley & Sons, p.167–202.

Wahle, K. (2000). Wie wird Software lokalisiert? [How is software localised?]. In: Schmitz, K.-D. and Wahle, K. (eds.), *Softwarelokalisierung [Software localisation]*, Tübingen, Germany: Stauffenburg, p.31–47.

Wasala, A., Schmidtke, D. and Schäler, R. (2012). XLIFF and LCX: A Comparison. *Localisation Focus*, 11 (1), p.67–79. [Online]. Available at: http://www.localisation.ie/oldwebsite/resources/lfresearch/Vol11_1WasalaSchmidtkeSchaler.htm [Accessed: 14 October 2015].

Weiss, A. and Evers, V. (2011). Exploring cultural factors in human-robot interaction: A matter of personality? In: *2nd International Workshop on Comparative Informatics*, 2011, Copenhagen, Denmark. [Online]. Available at: http://doc.utwente.nl/79601/ [Accessed: 14 October 2015].

White, N. (2012). Understanding the role of non-technical skills in patient safety. *Nursing Standard*, 26 (26), p.43–48.

Winter, J. and Rönkkö, K. (2010). SPI success factors within product usability evaluation. *Journal of Systems and Software*, 83 (11), p.2059–2072.

Wisker, G. (2008). *The Postgraduate Research Handbook*. 2nd ed. Basingstoke, UK: Palgrave Macmillan.

Wolff, F. (2006). Software localisation by Translate.org.za. *Localisation Focus*, 5 (3), p.19–21. [Online]. Available at: http://www.localisation.ie/oldwebsite/resources/lfresearch/Vol5_3Wolff.htm [Accessed: 14 October 2015].

Yao, J., Zhou, M., Zhao, T., Yu, H. and Li, S. (2002). An Automatic Evaluation Method for Localization Oriented Lexicalised EBMT System. In: *Second International Workshop on Language Resources for Translation Work, Research & Training*, 2002, Geneva, Switzerland.

Yuste, E. (2004). Corporate Language Resources in Multilingual Content Creation, Maintenance and Leverage. In: *Second International Workshop on Language Resources for Translation Work, Research & Training*, 2004, Geneva, Switzerland, p.9–15.

Yuste, E. (2005). Computer-aided technical translation workflows-man-machine in the construction and transfer of corporate knowledge. *Linguistik online*, 23 (2), p.67–75. [Online]. Available at: http://www.linguistik-online.de/23_05/yuste.pdf [Accessed: 14 October 2015].

Zahedi, F., Van Pelt, W. V. and Song, J. (2001). A conceptual framework for international web design. *IEEE Transactions on Professional Communication*, 44 (2), p.83–103.

Zerfaß, A. (2005). TMX - Austauschformat für Translation-Memory-Systeme [TMX - exchange format for translation memory systems]. In: Reineke, D. and Schmitz, K.-D. (eds.), *Einführung in die Softwarelokalisierung [Introduction to software localisation]*, Tübingen, Germany: Narr, p.169–175.

Zhang, C. Y. (2012). *The use of massively multiplayer online games to augment early-stage design process in construction*. PhD thesis, Loughborough, UK: Loughborough University. [Online]. Available at: https://dspace.lboro.ac.uk/2134/9924 [Accessed: 14 October 2015].

Zhang, P., Carey, J., Te'eni, D. and Tremaine, M. (2003). Integrating Human-Computer Interaction Development into the Systems Development Life Cycle: A Methodology. *Communications of the ACM*, 15, p.512–543.

Zhou, P. (2011). Managing the challenges of game localization. In: Dunne, K. J. and Dunne, E. S. (eds.), *Translation and Localization Project Management: The Art of the Possible*, American Translators Association Scholarly Monograph Series XVI, Amsterdam, The Netherlands: John Benjamins Pub. Co, p.349–378.

Zouncourides-Lull, A. (2011). Applying PMI methodology to translation and localization projects - Project Integration Management. In: Dunne, K. J. and Dunne, E. S. (eds.), *Translation and Localization Project Management: The Art of the Possible*, American Translators Association Scholarly Monograph Series XVI, Amsterdam, The Netherlands: John Benjamins Pub. Co, p.71–93.

Zuboff, S. (1988). *In the age of the smart machine: The future of work and power*. New York, NY, USA: Basic Books.

Credits

THE UNIVERSITY OF WEST LONDON PRESENTS A SOCIOTECHNICAL CENTRE FOR INTERNATIONALISATION AND USER EXPERIENCE PRODUCTION A SCHOOL OF COMPUTING AND ENGINEERING PRESENTATION

A MALTE REIJN THESIS "AN EMPIRICAL EXAMINATION OF INTERDISCIPLINARY COLLABORATION WITHIN THE PRACTICE OF LOCALISATION AND DEVELOPMENT OF INTERNATIONAL SOFTWARE"

PRINCIPAL SUPERVISOR Dr JOSÉ ABDELNOUR-NOCERA ADDITIONAL SUPERVISOR Dr STEPHEN ROBERTS THANKS FOR A GREAT TIME ALI GHEITASY ANN AUSTIN VAN AUL Dr JIVA BAGALE Dr ANTONIO DARLISH "YOU DON'T ASK THAT!" KHERKHAZADEH

Dr ANNA "WHY?" KOCUROVA Dr DEAN "NOT THE ROOM YOU ARE LOOKING FOR" KRAMER MOHAMMAD PUTTAROO Dr CHRISTIAN SAUER SHIVAN TOVI G FOR INVALUABLE ACADEMIC ADVICE Prof JOËLLE FANGHANEL

Prof PETER KOMSARCZUK Dr DIETER LEGNER Prof KATHRYN MITCHELL Dr LAURA MORGANTINI Prof DIMITRIOS RIGAS Dr JANNE ROED LITERATURE PROVISION SUNNY BIRD EDYTA KROL & THE UWL LIBRARIANS

ADDITIONAL LITERATURE Dr SAMEER ABUFARDEH KATIE BOTKIN Prof CHRISTIAN STURM Dr HUATONG SUN TECHNICAL SUPPORT NEIL HAMMOND ADMINISTRATIVE SUPPORT MARIA PENNELLS SURVEY ASSISTING CENTRE FOR MODEL-DRIVEN SOFTWARE ENGINEERING

EXTERNAL EXAMINER Prof PAT HALL INTERNAL EXAMINER Dr SAMA OUSSENA EXAMINATION CHAIR Dr ALI BAHADORI-JAHROMI SHELTER & CATERING HONEYCRUMBS BAKERY ISLEWORTH PASTORAL SUPPORT PREA TARAN LIARA JUTTA & BENNO IN MEMORIAM Prof ANDY SMITH

INEVITABLE WISDOM FIGURING THINGS OUT FOR OURSELVES AND LOOKING BEYOND WHAT IS IN FRONT OF OUR NOSE IS THE GREATEST FREEDOM WE HAVE

SPECIAL THANKS ALL INTERVIEW AND SURVEY PARTICIPANTS AND EVERYBODY WHO HELPED ME FIND THEM